

Учебное пособие по программной отладке на C++ и Java

Авторы: Астанин Анатолий, Дочкин Никита, Крюкова Ксения

Введение

В настоящее время программирование является динамично развивающейся областью, и каждый человек, который относит себя к этой области, должен соответствовать этому развитию.

Процесс отладки программы занимает большую часть времени работы программиста. Данный процесс занимает ещё больше времени в том случае, если вы не используете программные средства для этого.

Программная отладка является одним из важных инструментов, которым обязан уметь пользоваться любой программист. Каждый программист занимается отладкой своих программ, но не каждый делает это с помощью специальных средств.

Наша работа направлена на то, чтобы ознакомить программистов, не использующих средства отладки, с данным полезнейшим инструментом.

На простых примерах с подробным описанием шагов вы овладеете начальными навыками отладки программ на языках C++ и Java. Вы сможете наглядно понять принципы и причины применения отладки. Ознакомившись с данным пособием, вы приобретёте не только начальные, но и немного продвинутых навыков отладки, таких как «Условия и действия точки останова»

Для кого данное пособие?

Данное пособие предназначено для программистов начинающего уровня, которые только начинают свой путь в сфере разработки и информационных технологий.

Также данное пособие будет полезно разработчикам, не знакомым с данным инструментом, студентам первых курсов ВУЗов и учреждений СПО, с направлениями подготовки, включающими в себя обучение языкам программирования C++ или Java.

Содержание

Введение.....	2
Для кого данное пособие?	2
Введение в отладку	4
Что такое отладка?	4
Место отладки в цикле разработки программного обеспечения	4
Терминологию и понятия, используемые в данном пособии.....	5
Введение в практическую отладку	6
Анализ проблемы	6
Типы ошибок	9
Теория отладки на практике	10
Примеры отладочных случаев	11
Отладка в среде программирования Microsoft Visual Studio на языке C++....	12
Степпинг	12
Точки останова	22
Отслеживание переменных	24
Окно просмотра.....	29
Стек вызовов.....	32
Действия точки останова и точки трассировки. Условия точки останова...	36
Задание точки останова функции.....	42
Управление точками останова в окне "Точки останова"	45
Задание точки останова в окне стека вызовов	47
Отладка в IDE IntelliJ IDEA на языке программирования Java.....	48
Дебагинг в IntelliJ IDEA	48
Степпинг	50
Точки останова	59
Точки останова с условием	64
Заключение	68

Введение в отладку

Если вы уже знаете, что такое отладка, то переходите к следующей главе. В этой главе вы узнаете:

- Что такое отладка?
- Место отладки в цикле разработки программного обеспечения
- Терминологию и понятия, используемые в данном пособии

Что такое отладка?

Отладка — этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки. Чтобы понять, где возникла ошибка, приходится: узнавать текущие значения переменных; выяснять, по какому пути выполнялась программа.

Использование отладчиков — программ, которые включают в себя пользовательский интерфейс для пошагового выполнения программы: оператор за оператором, функция за функцией, с остановками на некоторых строках исходного кода или при достижении определённого условия.

Место отладки в цикле разработки программного обеспечения

Типичный цикл разработки, за время жизни программы многократно повторяющийся, выглядит примерно так:

1. Программирование — внесение в программу новой функциональности, исправление существующих ошибок.
2. Тестирование — обнаружение факта ошибки.
3. Воспроизведение ошибки — выяснение условий, при которых ошибка случается. Это может оказаться непростой задачей при программировании параллельных процессов и при некоторых необычных ошибках, известных как гейзенбаги.

4. Отладка — обнаружение причины ошибки.

Терминологию и понятия, используемые в данном пособии

- Отладка (дебаг, дебаггинг) - процесс нахождения и исправления ошибок в исходном коде программы. Процесс поиска, локализации и исправления ошибок в компьютерной программе. На всех этапах разработки программного обеспечения О. п. тесно связана с тестированием программ. В процессе тестирования (напр., с использованием спец. подобранных наборов исходных данных) могут достигаться такие состояния программы, в которых фиксируются расхождения с заданными спецификациями. О. п. обеспечивает поиск причин этих расхождений (локализацию ошибок) и соответствующую корректировку программы. В процессе трансляции и выполнения программы (которая реализует некоторую функцию) компьютер, используя свои аппаратно-программные ресурсы, доопределяет (напр., вызывает из библиотеки стандартные подпрограммы) частично определённую программой функцию до т. н. тотально определённой. Т. о. судить о правильности или неправильности результатов выполнения программы можно только сравнивая заданную спецификацию функции с результатами её вычислений, что и осуществляется в процессах тестирования и отладки.

- Точка останова (далее брэкпойнт, точка останова) - точка, в которой программа прерывает своё выполнение до дальнейших действий программиста. Это специальные маркеры, на которых отладчик останавливает процесс выполнения программы.

- Степпинг (англ. «stepping») — это возможность отладчика выполнять код пошагово (строка за строкой).

Введение в практическую отладку

Если вы уже знаете, как анализировать ошибки, типы ошибок и как производить отладку на практике, то переходите к следующей статье. В этой главе вы узнаете:

- Анализ проблемы
- Типы ошибок
- Теория отладки на практике
- Примеры отладочных случаев

Анализ проблемы

Для начала, в процессе написания программы, в случае возникновения ошибок, вызывающих неожиданное поведение алгоритмов и/или блоков кода, необходимо задать себе два вопроса:

- Что должно было произойти?
- Что пошло не так?

Чтобы ответить на первый вопрос, мы должны иметь данные успешно работающего алгоритма и/или эксперимента. На данный вопрос, как правило, просто ответить.

Второй вопрос обычно требует как минимум рассмотрения кода на предмет ошибок. Здесь нам и понадобится отладчик. Во многих случаях возникает ситуация, в которой простого рассмотрения кода и вывода отладочных сообщений будет либо недостаточно, либо это займет слишком много времени, по сравнению с программной отладкой. Имея средство отладки и профилирования программы, программист может с легкостью отследить путь её выполнения, значения переменных в любой момент времени, выполнить программу шаг за шагом. Всё это позволяет с легкостью определить ошибки в коде.

Предположите, что именно в вашей программе пошло не так, в чём может заключаться непредвиденное поведение? Проверьте ваши предположения:

- Используете ли вы нужный API (то есть соответствующие объект, функцию, метод или свойство)? Возможно, используемый вами API работает не так, как вы ожидаете. (После проверки вызова API в отладчике для исправления проблемы и выявления нужного API вам может потребоваться обратиться к документации.)
- Правильно ли вы используете API? Даже если вы выбрали нужный API, он может использоваться неправильно.
- Нет ли в вашем коде опечаток? Некоторые опечатки, например ошибки в написании имени переменной, могут быть незаметными, особенно при работе с языками, в которых не требуется объявление переменных перед их использованием.
- Вносили ли вы изменения в код и могут ли они быть связаны с возникшей проблемой?
- Должны ли объект или переменная содержать определенное значение (или определенный тип значения) и соответствует ли это действительности?
- Известно ли назначение кода? Как правило, отладка кода, написанного другим разработчиком, дается гораздо сложнее. Если это не ваш код, возможно, для его эффективной отладки вам потребуется изучить, что он делает.

Стоит отметить, что отладка не является "магическим" инструментом, который найдет и исправит все ошибки за вас. Одного отладчика для нахождения неточностей недостаточно. Кроме отладчика, программисту нужно понимание того, что написано в его оде, понимание того, как это работает, чтобы определить, в каком примерно участке программы возникла ошибка.

После анализа проблемы и нахождения конкретного места ошибки мы можем приступить к дебагу, то есть к процессу исправления кода.

В случаях, когда проделать анализ, поиск и дебаг недостаточно для полного исправления "ошибочной" программы, может помочь повторное пошаговое воспроизведение. Если и этого будет недостаточно, то скорее всего, ошибка не в коде программы, а в алгоритмах, обрабатывающих информацию.

Типы ошибок

Синтаксическая ошибка возникает, когда вы пишете код, который не соответствует правилам грамматики языка C++. Например, пропущенные точки с запятой, необъявленные переменные, непарные круглые или фигурные скобки и т.д.

Семантическая ошибка возникает, когда код является синтаксически правильным, но делает не то, что задумал программист.

Теория отладки на практике

Прежде, чем приступить к практике, стоит разобраться в основном алгоритме отладки. Выше мы уже ознакомили вас с основными понятиями, которые необходимы для понимания процесса.

1. Запустить программу
2. Проверить программу на ошибки, в случае, если их нет перейти к пункту 8
3. Проанализировать ошибку/баг/неявное поведение. Выдвинуть предположение (что именно пошло не так? в какой части кода может быть ошибка? что-то не так с кодом?)
4. Расставить точки останова(см.далее) в местах, где по вашему мнению содержится код, который необходимо исправить.
5. Запустить отладочную версию программы (Debug)
6. Пошагово выполнить фрагмент кода
7. В случае обнаружения ошибки/бага исправить его, если ошибку не нашли либо вернуться к пункту 6 и выполнить код на других данных, либо вернуться к пункту 3
8. Завершить отладку программы

Примеры отладочных случаев

Зачастую на стадии компиляции компилятор и так укажет вам на синтаксические ошибки. Но что будет, если синтаксически всё будет написано правильно? Допустим, вы забыли где-то скобки (например, в вычислениях) или взяли другую переменную в качестве параметра. Безусловно, в такой ситуации компилятор скомпилирует вашу программу, но изначально ошибку вы не увидите. Теперь представьте что ваш проект состоит из нескольких тысяч или сотен тысяч строк кода. Представили? Теперь представьте что в одной из строк пропущена скобка. Вряд-ли вы будете сидеть и глазами искать её. Для таких случаев и существует отладчик.

Отладка программы позволит вам пошагово отследить, в какой момент выполнение начинает идти по отклоняющейся траектории. Соответственно, именно в тех строках, на которых, говоря простым языком, "спотыкается" программа, и находится ошибка. Благодаря отладке вы смогли в разы сократить время на поиск неточностей, и теперь можете смело редактировать их. Поздравляем, вы великолепны!

Иногда отладка помогает обнаружить опечатки, наличие которых в корне меняет суть всей работы. В этом плюс именно пошагового выполнения с указанием исполняемой строки. Например, в случае если выполняется функция, схожая по названию с требуемой (команда проекта настоятельно не рекомендует называть функции и переменные практически идентично, если того не требует регламент), вы можете отследить это не только в одном конкретном месте, но и во всём коде.

Настоятельно рекомендуем вам выдвигать предположения о том, что могло пойти не так. Понимание кода - один из ключевых навыков программиста.

Отладка в среде программирования Microsoft Visual Studio на языке C++

Степпинг

Если вы уже знакомы с пошаговым выполнением программ, то переходите к следующей под-главе.

В данной главе вы узнаете о:

- Степпинге
- Точках останова

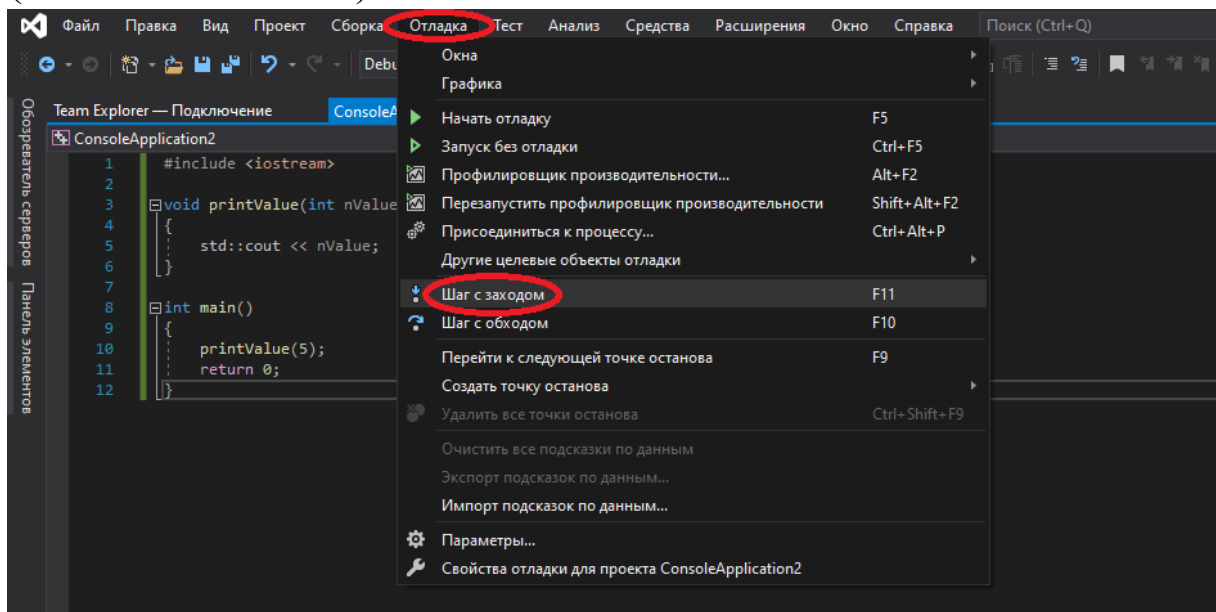
Пример исходного кода можете найти в репозитории, перейдя по ссылке « <https://github.com/12PAIN/UchPrakt/Examples/C++> » .
Название файла: Example_1.cpp

▪ Шаг с заходом

Команда «Шаг с заходом» (англ. «Step into») выполняет следующую строку кода. Если этой строкой является вызов функции, то «Шаг с заходом» открывает функцию и выполнение переносится в начало этой функции.

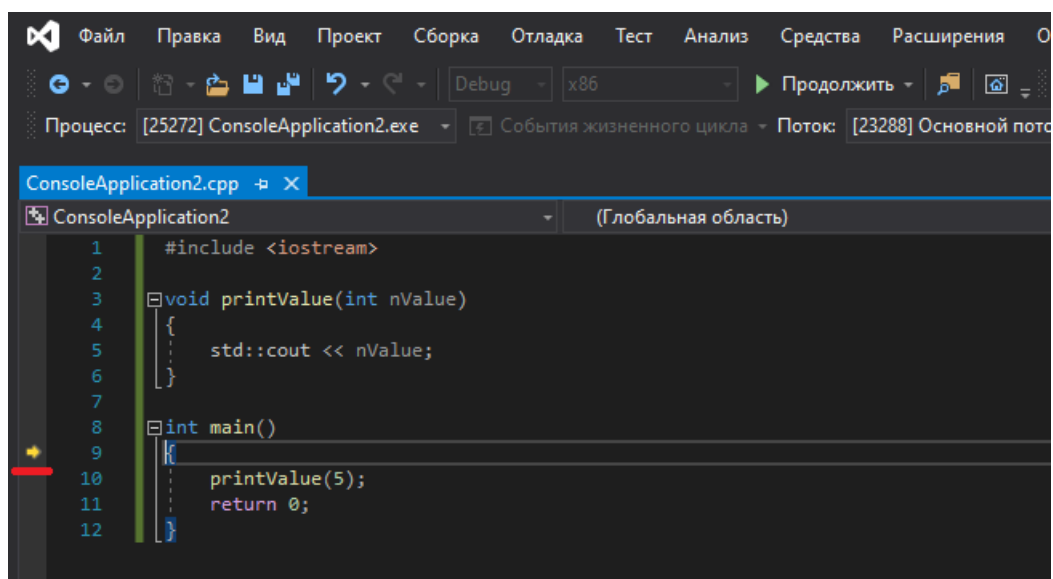
Как вы уже знаете, при запуске программы выполнение начинается с вызова главной функции `main()`. Так как мы хотим выполнить отладку внутри функции `main()`, то давайте начнем с использования команды «Шаг с заходом».

В Visual Studio, перейдите в меню "Отладка" → "Шаг с заходом" (либо нажмите F11):

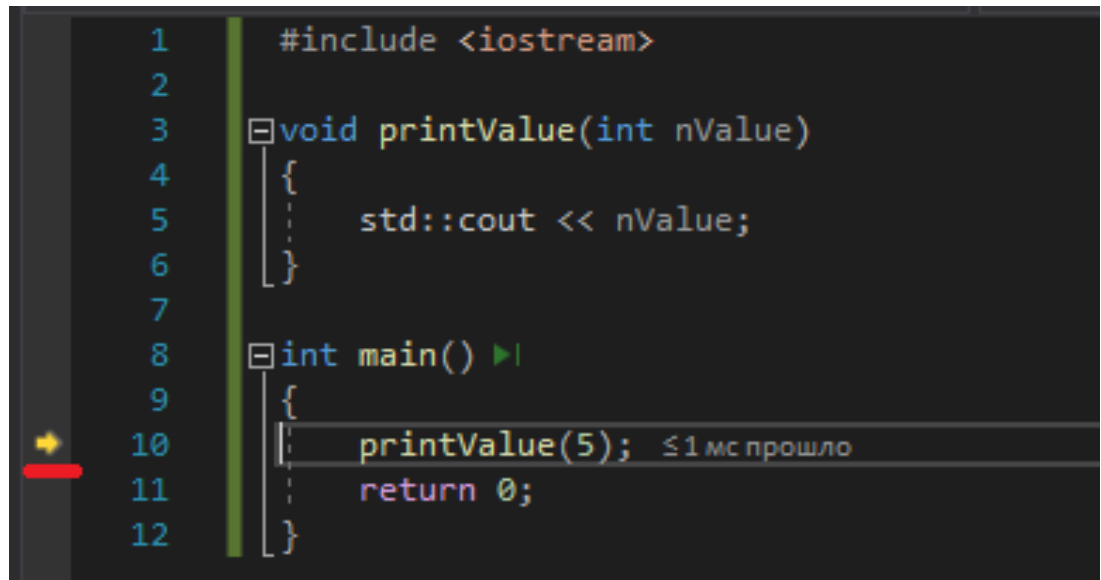


Если вы используете другую IDE, то найдите в меню команду "Step Into/Шаг с заходом" и выберите её.

Когда вы это сделаете, должны произойти две вещи. Во-первых, так как наше приложение является консольной программой, то должно открыться консольное окно. Оно будет пустым, так как мы еще ничего не выводили. Во-вторых, вы должны увидеть специальный маркер слева возле открывающей скобки функции main(). В Visual Studio этим маркером является жёлтая стрелочка (если вы используете другую IDE, то должно появиться что-нибудь похожее):

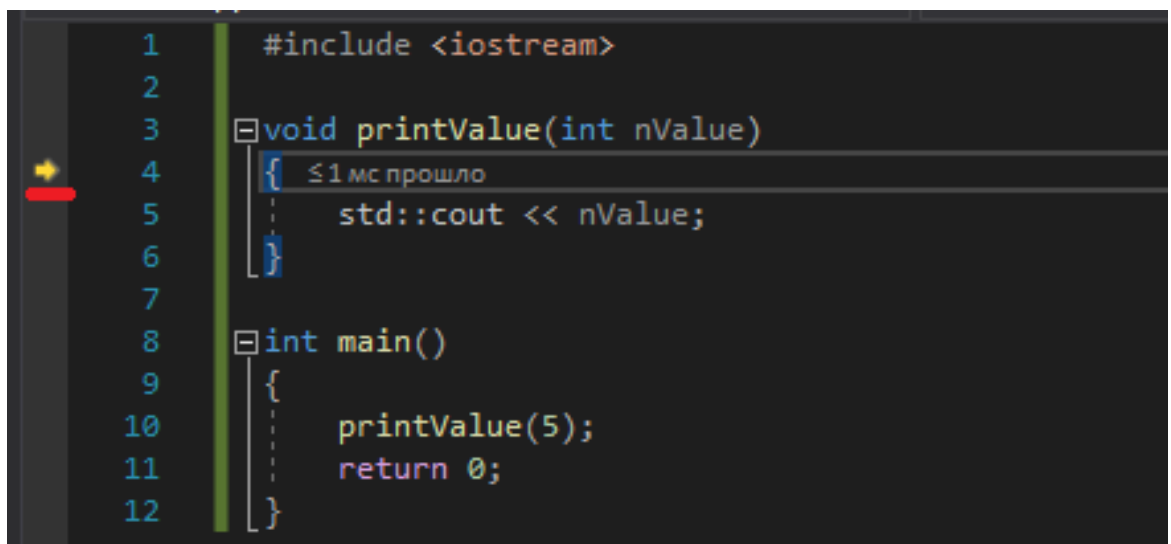


Стрелка-маркер указывает на следующую строку, которая будет выполняться. В этом случае отладчик говорит нам, что следующей строкой, которая будет выполняться, — будет открывающая фигурная скобка функции `main()`. Выберите «Шаг с заходом» еще раз — стрелка переместится на следующую строку:



```
1  #include <iostream>
2
3  void printValue(int nValue)
4  {
5      std::cout << nValue;
6  }
7
8  int main()
9  {
10     printValue(5);
11     return 0;
12 }
```

Это значит, что следующей строкой, которая будет выполняться, — будет вызов функции `printValue()`. Выберите «Шаг с заходом» еще раз. Поскольку `printValue()` — это вызов функции, то мы переместимся в начало функции `printValue()`:

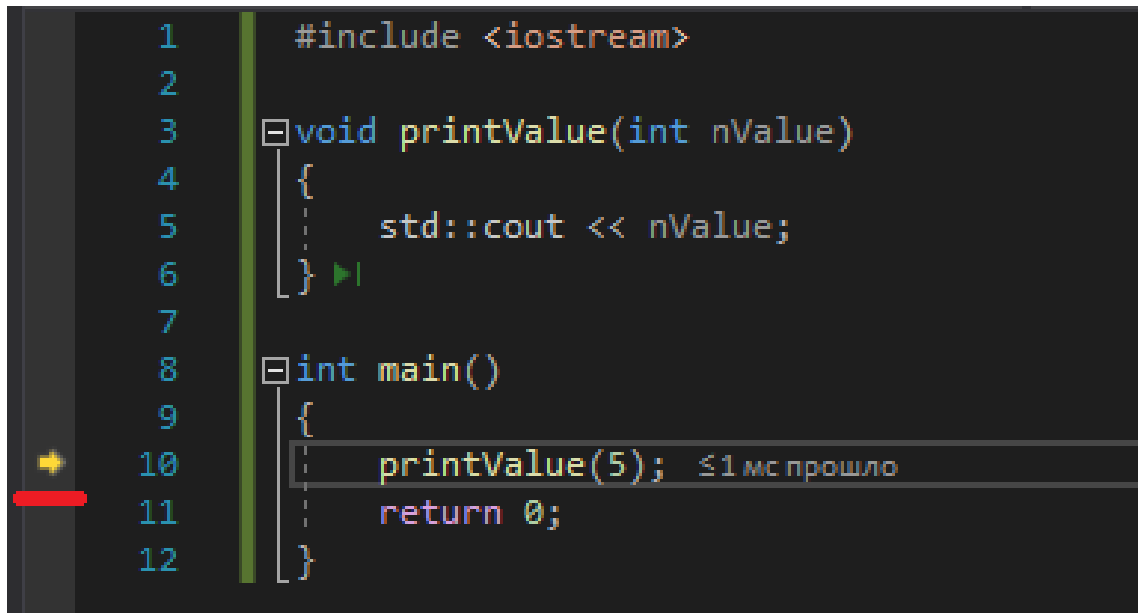


```
1  #include <iostream>
2
3  void printValue(int nValue)
4  {
5      std::cout << nValue;
6  }
7
8  int main()
9  {
10     printValue(5);
11     return 0;
12 }
```

Выберите еще раз «Шаг с заходом» для выполнения открывающей фигурной скобки `printValue()`. Стрелка будет указывать на `std::cout << nValue;`.

Теперь выберите «Шаг с обходом» (F10). Вы увидите число 5 в консольном окне.

Выберите «Шаг с заходом» еще раз для выполнения закрывающей фигурной скобки `printValue()`. Функция `printValue()` завершит свое выполнение и стрелка переместиться в функцию `main()`. Обратите внимание, в `main()` стрелка снова будет указывать на вызов `printValue()`:

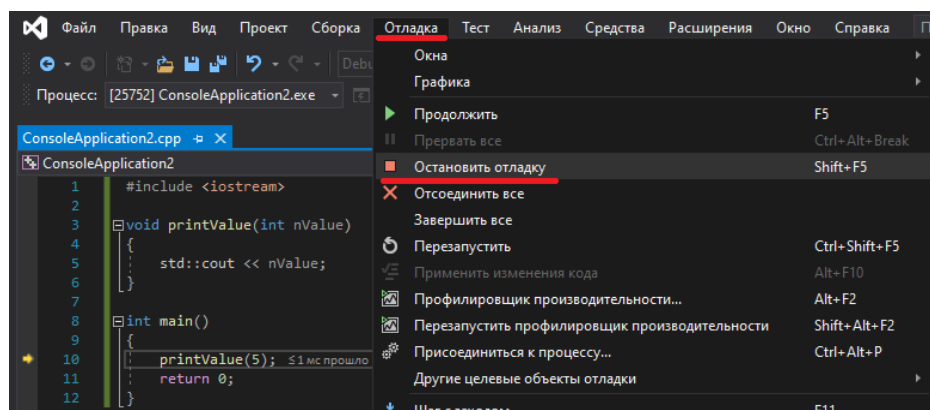


```
1  #include <iostream>
2
3  void printValue(int nValue)
4  {
5      std::cout << nValue;
6  }
7
8  int main()
9  {
10     printValue(5);
11     return 0;
12 }
```

The screenshot shows a C++ program in Visual Studio. A red arrow points to line 10, which is `printValue(5);`. A tooltip above the line indicates that the function call took less than 1 ms (`≤1 мс прошло`).

Может показаться, будто отладчик намеревается еще раз повторить цикл с функцией `printValue()`, но в действительности он нам просто сообщает, что он только что вернулся из этой функции.

Выберите «Шаг с заходом» два раза. Готово, все строки кода выполнены. Некоторые дебаггеры автоматически прекращают сеанс отладки в этой точке. Но Visual Studio так не делает, так что если вы используете Visual Studio, то выберите "Отладка" → "Остановить отладку" (или Shift+F5)



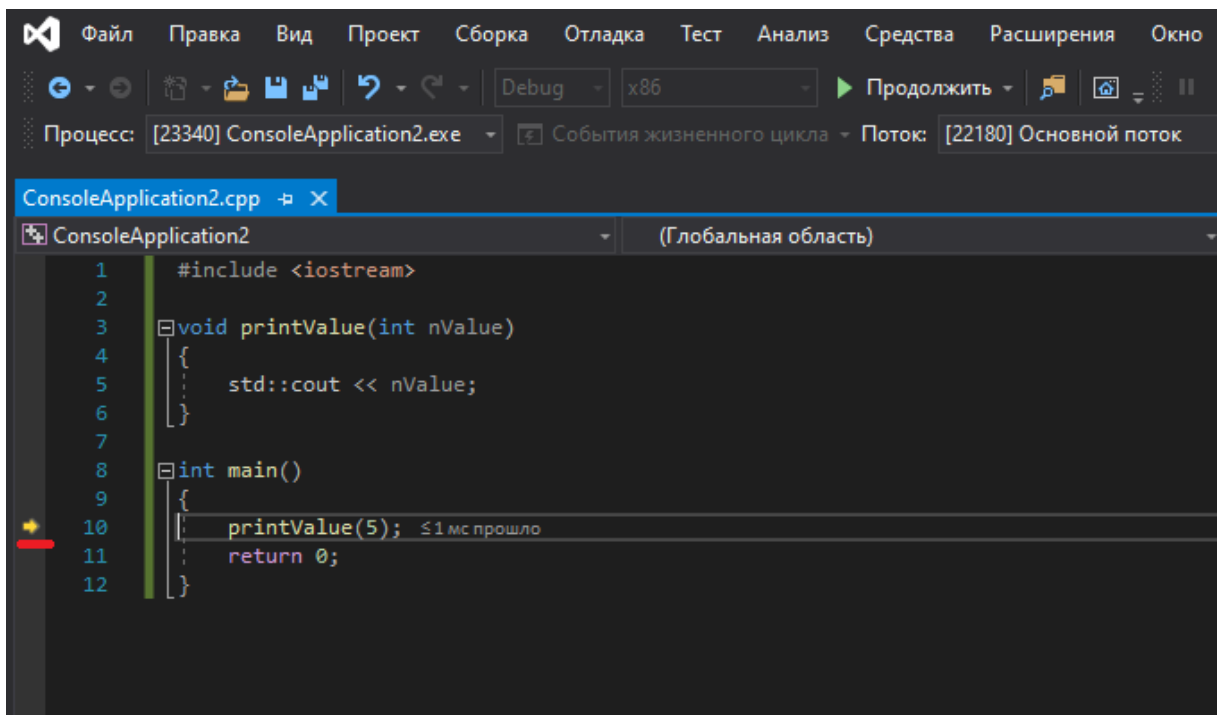
Таким образом мы полностью остановили сеанс отладки нашей программы.

▪ Шаг с обходом

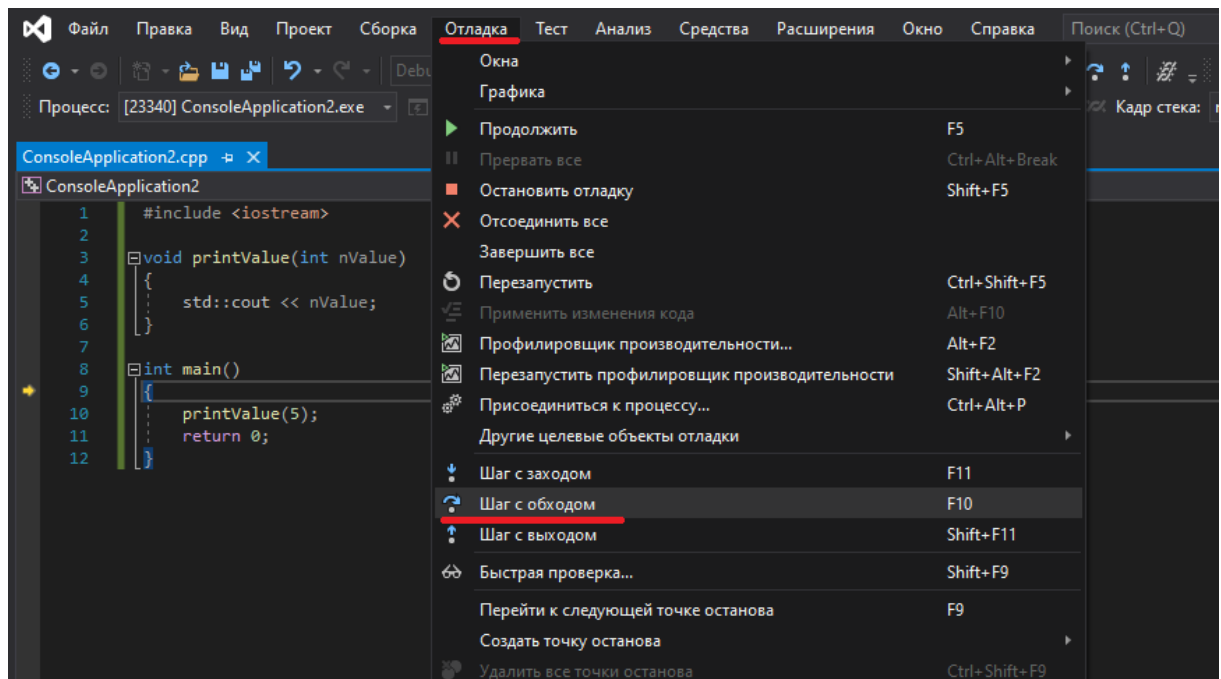
Команда «Шаг с обходом» Как и команда «Шаг с заходом», команда «Шаг с обходом» (англ. «Step over») позволяет выполнить следующую строку кода. Только если этой строкой является вызов функции, то «Шаг с обходом» выполнит весь код функции в одно нажатие и возвратит нам контроль после того, как функция будет выполнена.

Рассмотрим пример, используя ту же программу.

Нажмите «Шаг с заходом», чтобы дойти до вызова функции `printValue()`:



Теперь вместо команды «Шаг с заходом» выберите «Шаг с обходом» (или F10):



Отладчик выполнит функцию (которая выведет значение 5 в консоль), а затем возвратит нам управление на строке `return 0;`. И это всё за одно нажатие.

Команда «Шаг с обходом» позволяет быстро пропустить код функций, когда мы уверены, что они работают корректно и их не нужно отлаживать.

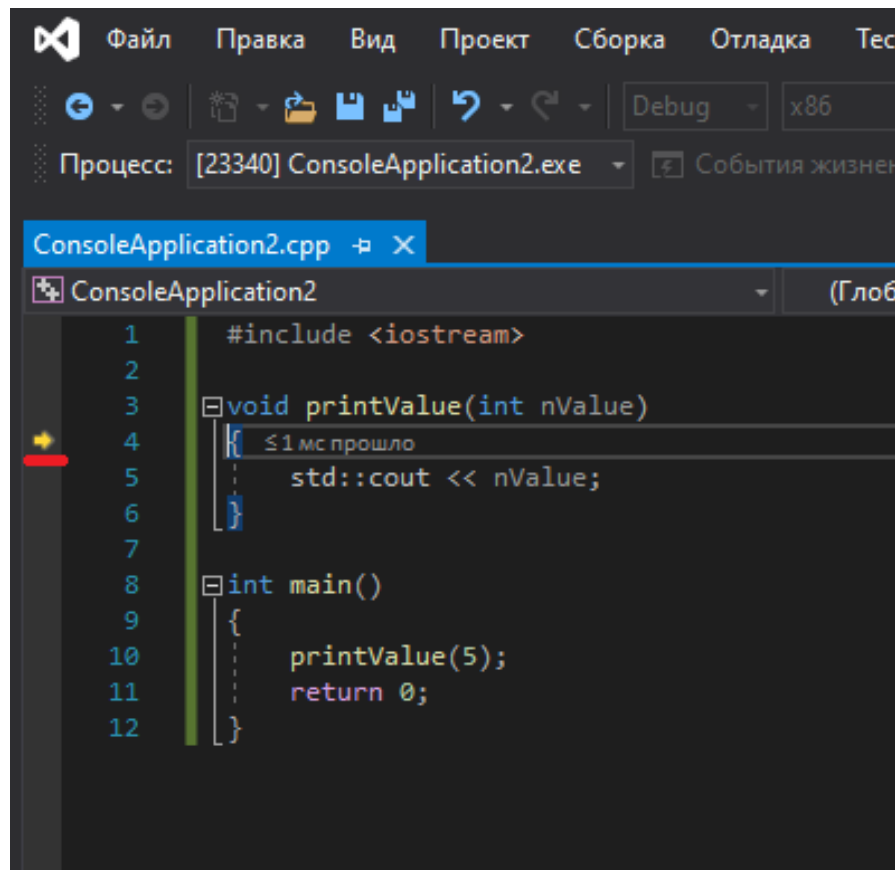
■ Шаг с выходом

В отличие от двух предыдущих команд, команда «Шаг с выходом» (англ. «Step out») не просто выполняет следующую строку кода. Она выполняет весь оставшийся код функции, в которой вы сейчас находитесь, и возвращает контроль только после того, когда функция завершит свое выполнение. Проще говоря, «Шаг с выходом» позволяет выйти из функции.

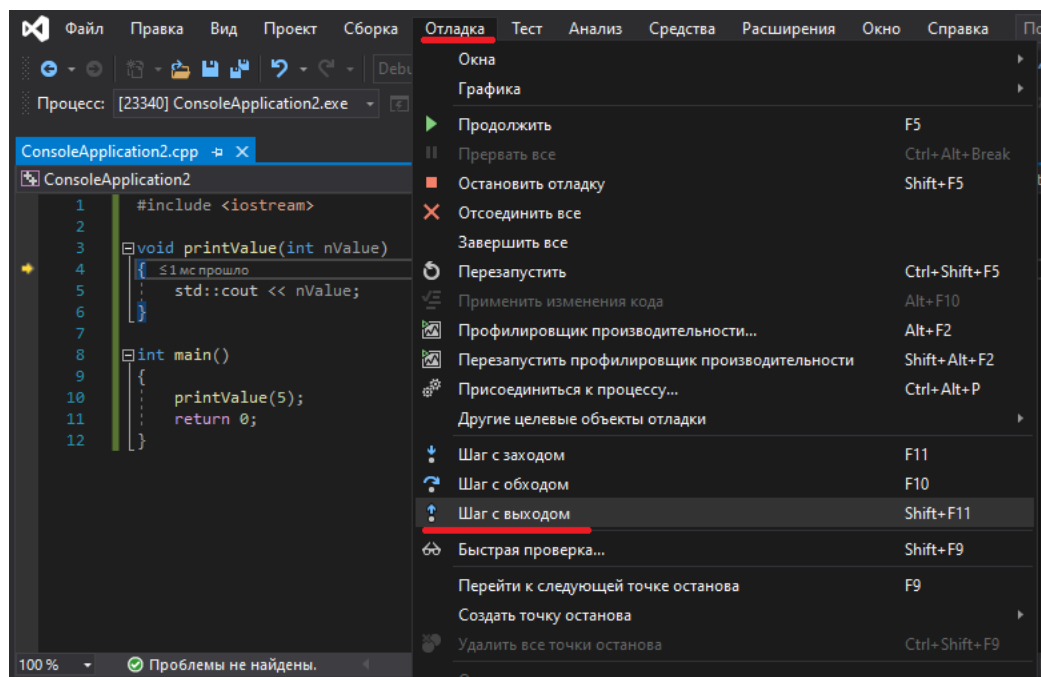
Обратите внимание, команда «Шаг с выходом» появится в меню «Отладка» только после начала сеанса отладки (что делается путем использования одной из двух вышеприведенных команд).

Рассмотрим все тот же пример.

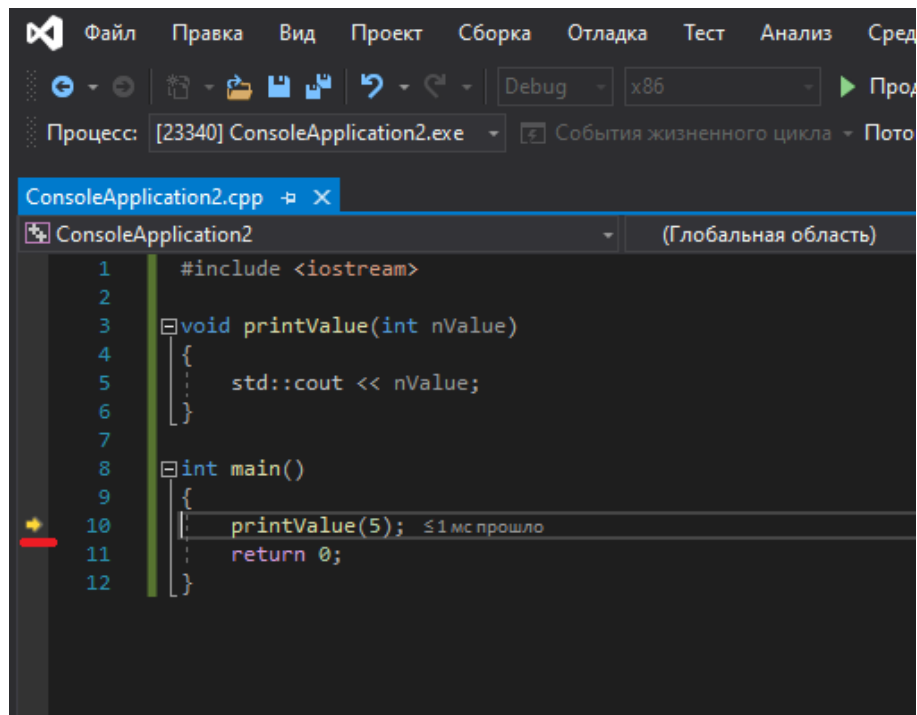
Нажимайте «Шаг с заходом» до тех пор, пока не перейдете к открывающей фигурной скобке функции `printValue()`:



Затем выберите "Отладка" → "Шаг с выходом" (либо Shift+F11):



Вы заметите, что значение 5 отобразилось в консольном окне, а отладчик перешел к вызову функции printValue() в main():



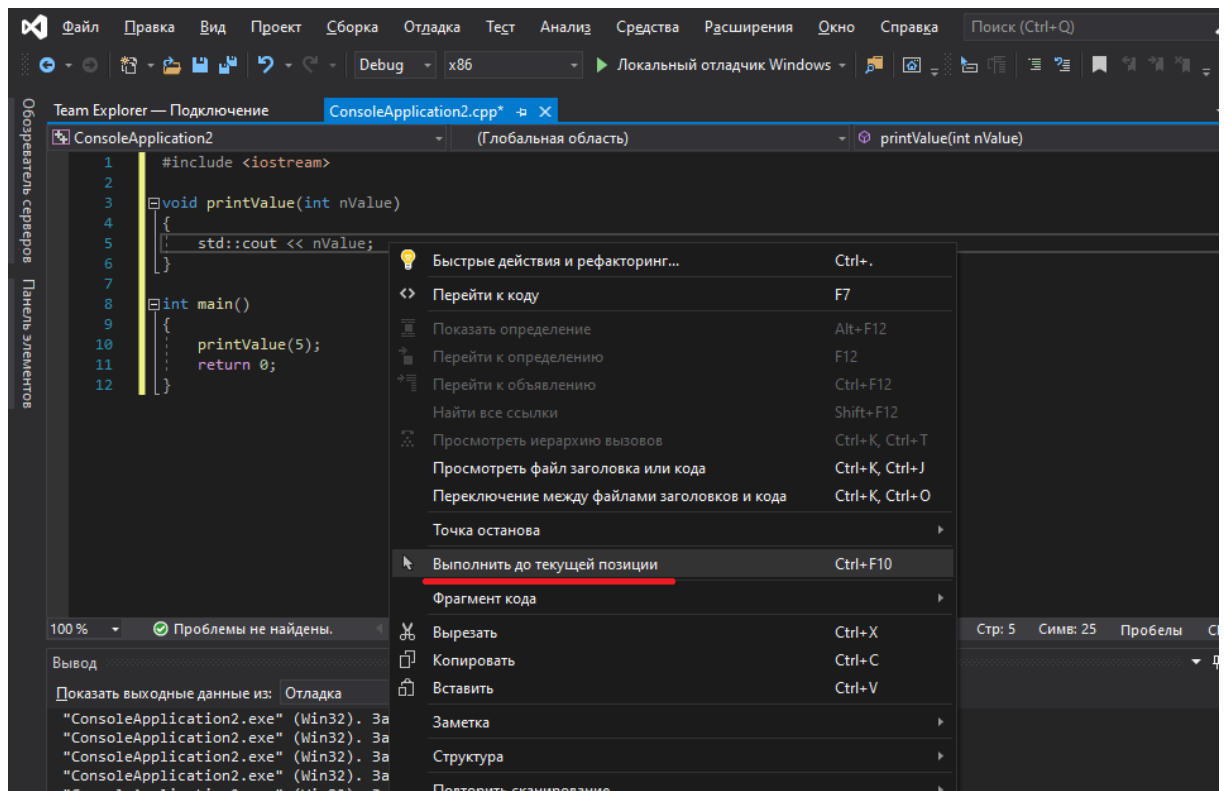
▪ Выполнить до текущей позиции

В то время как степпинг полезен для изучения каждой строки кода по отдельности, в большой программе перемещаться по коду с помощью этих команд будет не очень удобно.

Но и здесь современные отладчики предлагают еще несколько инструментов для эффективной отладки программ.

Команда «Выполнить до текущей позиции» позволяет в одно нажатие выполнить весь код до строки, обозначенной курсором. Затем контроль обратно возвращается к вам, и вы можете проводить отладку с указанной точки уже более детально.

Поместите курсор на строку `std::cout << nValue;` внутри функции `printValue()`, затем щелкните правой кнопкой мыши и выберите "Выполнить до текущей позиции" (либо `Ctrl+F10`):

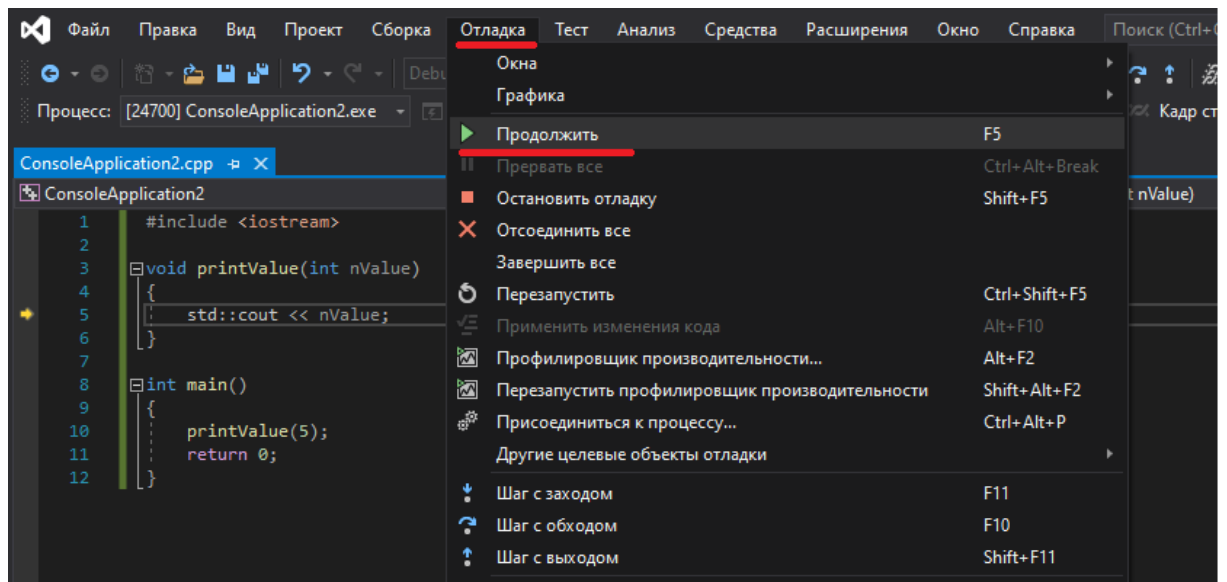


Вы заметите, что жёлтая стрелочка переместится на указанную вами строку. Выполнение программы остановится в этой точке, и программа будет ждать ваших дальнейших команд.

▪ Продолжить

Если вы находитесь в середине сеанса отладки вашей программы, то вы можете сообщить отладчику продолжать выполнение кода до тех пор, пока он не дойдет до конца программы (или до следующей контрольной точки). В Visual Studio эта команда называется «Продолжить» (англ. «Continue»). В других дебаггерах она может иметь название «Run» или «Go».

Возвращаясь к вышеприведенному примеру, мы находимся как раз внутри функции `printValue()`. Выберите "Отладка" → "Продолжить" (или F5):



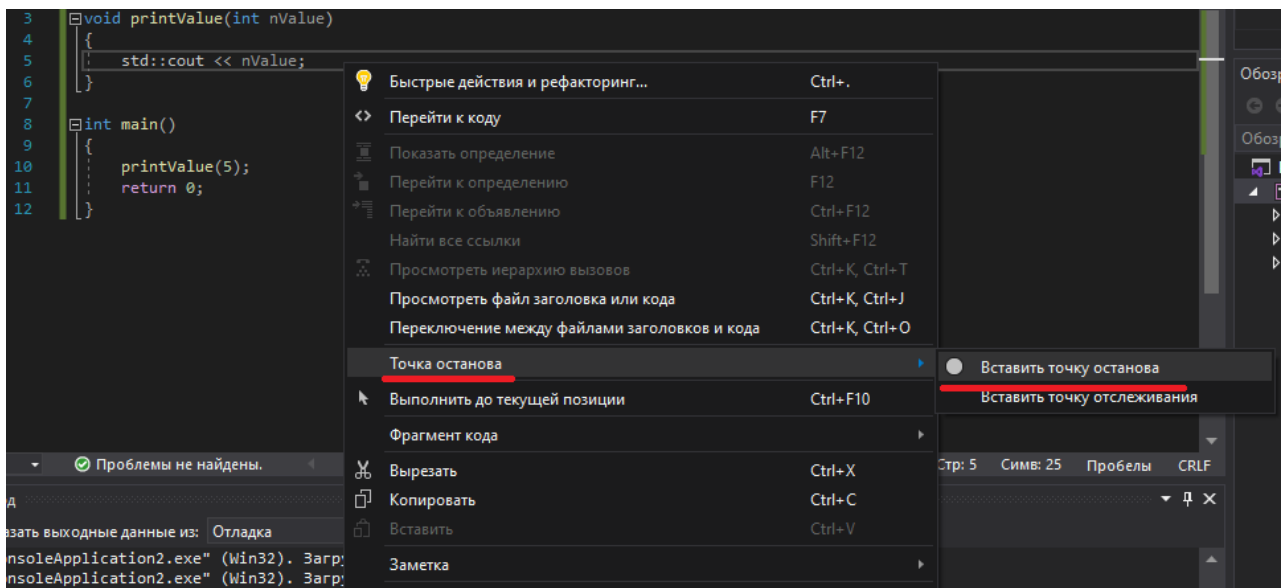
Программа завершит свое выполнение и выйдет из сеанса отладки.

Точки останова

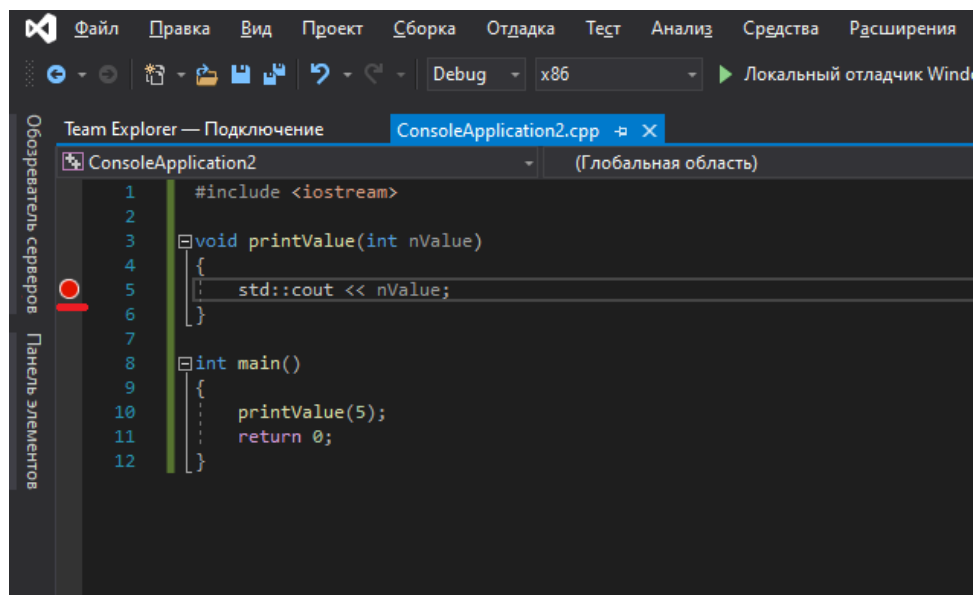
Точки останова (англ. «breakpoints») — это специальные маркеры, на которых отладчик останавливает процесс выполнения программы.

Если вы уже знаете, как пользоваться точками останова, то переходите к следующей под-главе.

Чтобы задать точку останова в Visual Studio, щелкните правой кнопкой мыши по выбранной строке → "Точка останова" → "Вставить точку останова":



Появится кружок возле строки, в которую вы вставили точку останова:



Точки останова чрезвычайно полезны, если вы хотите изучить только определенную часть кода. Просто задайте точку останова в выбранном участке кода, выберите команду «Продолжить» и отладчик автоматически остановится возле указанной строки. Затем вы сможете использовать команды степпинга для более детального просмотра/изучения кода.

Отслеживание переменных

Отслеживание переменных – возможность посмотреть и проверить значения переменных, используемых в программе во время отладки.

Пример исходного кода можете найти в репозитории, перейдя по ссылке «
<https://github.com/12PAIN/UchPrakt/tree/main/Examples/C%2B%2B> ».
Название файла: Example_2.cpp

Исходный код:

```
#include <iostream>

int main()
{
    int x = 1;

    std::cout << x << " ";

    x = x + 1;

    std::cout << x << " ";

    x = x + 2;

    std::cout << x << " ";

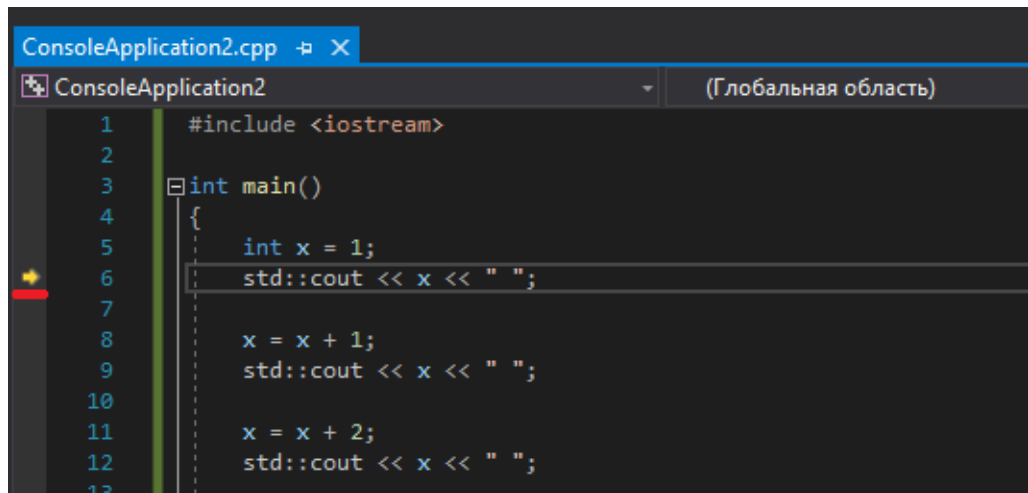
    x = x + 4;

    std::cout << x << " ";

    return 0;
}
```

Результатом выполнения программы будет такая строка: 1 2 4 8

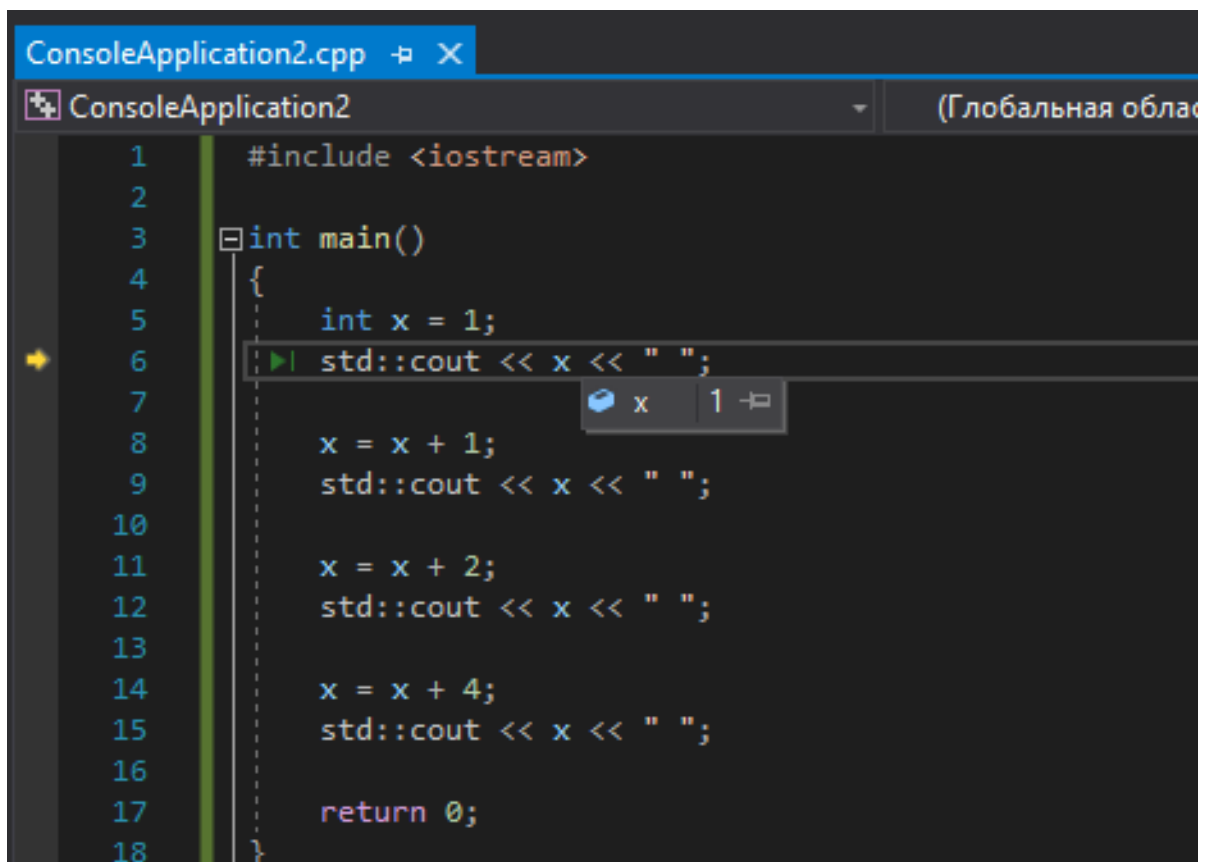
Используя команду «Выполнить до текущей позиции» вы переместитесь к строке `std::cout << x << " "`;



```
ConsoleApplication2.cpp X
ConsoleApplication2 (Глобальная область)
1  #include <iostream>
2
3  int main()
4  {
5      int x = 1;
6      std::cout << x << " ";
7
8      x = x + 1;
9      std::cout << x << " ";
10
11     x = x + 2;
12     std::cout << x << " ";
13 }
```

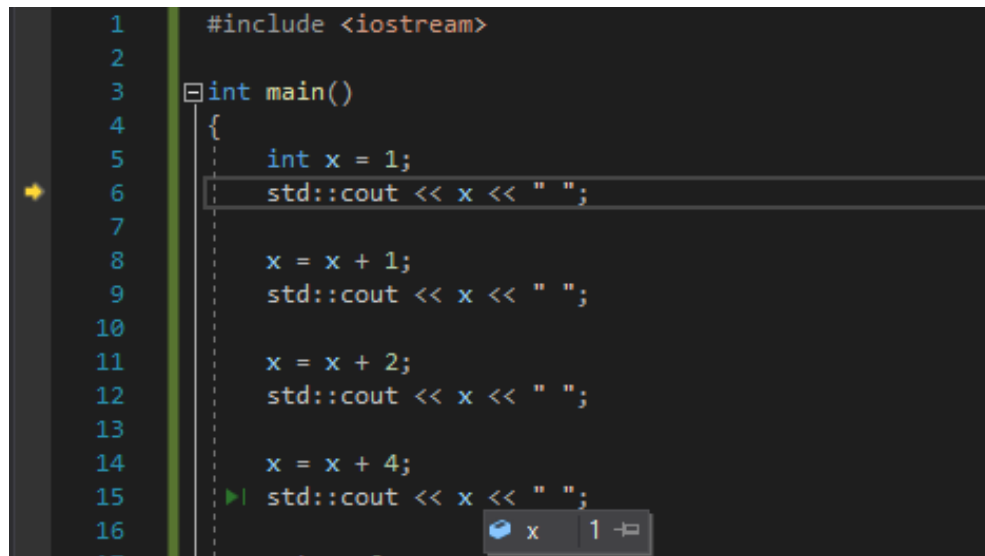
В момент времени, когда вы это сделаете, переменная с названием `x` уже создана и инициализирована

Самый быстрый способ посмотреть значение обычной переменной (таких как `x`) – навести курсор мыши на элемент. Большая часть отладчиков поддерживает эту функцию:



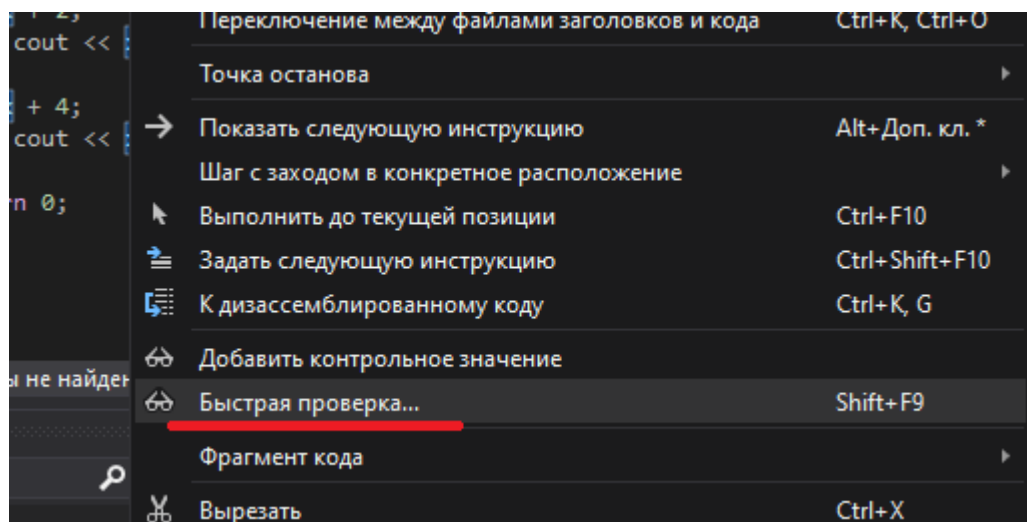
```
ConsoleApplication2.cpp X
ConsoleApplication2 (Глобальная область)
1  #include <iostream>
2
3  int main()
4  {
5      int x = 1;
6      std::cout << x << " ";
7
8      x = x + 1;
9      std::cout << x << " ";
10
11     x = x + 2;
12     std::cout << x << " ";
13
14     x = x + 4;
15     std::cout << x << " ";
16
17     return 0;
18 }
```

Также вы можете сделать это на любой другой переменной или строчке:

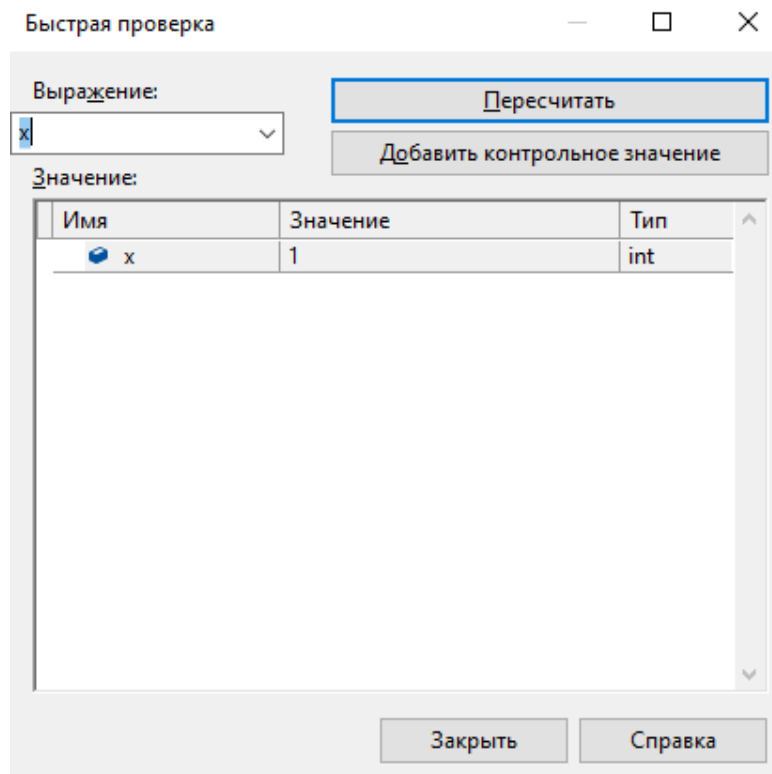


```
1  #include <iostream>
2
3  int main()
4  {
5      int x = 1;
6      std::cout << x << " ";
7
8      x = x + 1;
9      std::cout << x << " ";
10
11     x = x + 2;
12     std::cout << x << " ";
13
14     x = x + 4;
15     std::cout << x << " ";
16 }
```

В IDE Microsoft Visual Studio есть функция «Быстрой проверки». Выделите переменную `x` и с помощью правой кнопки мыши откройте контекстное меню и выберите пункт «Быстрая проверка...»:

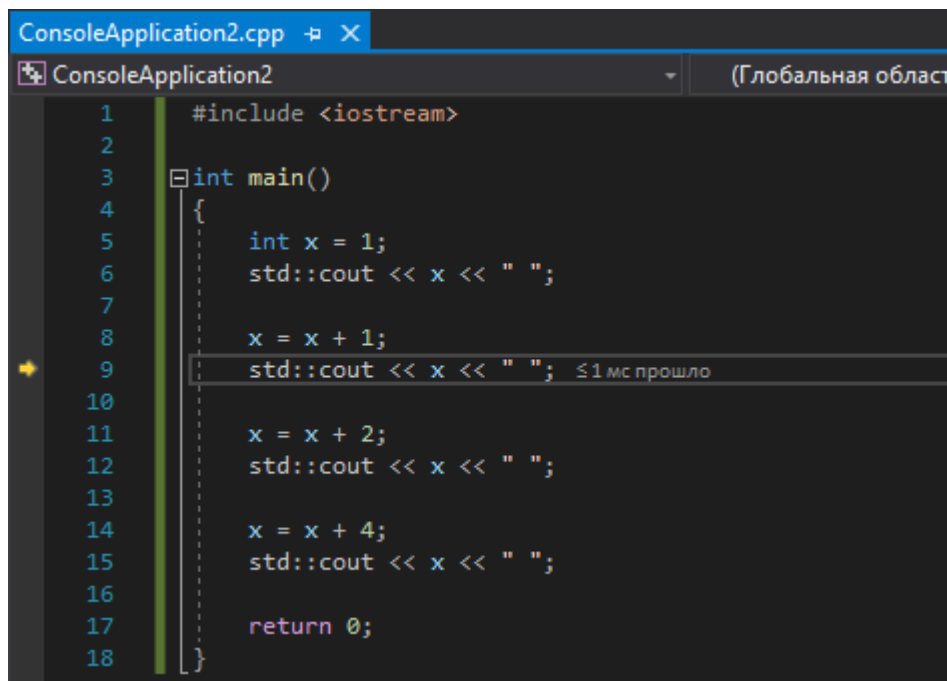


Тогда у вас откроется специальное окно со значением переменной:



Теперь вы можете закрыть это окно.

Также пользоваться функцией проверки значений можно и во время проведения отладки. Переместитесь по выполняемой программе, используя «Шаг с обходом»:



Наша переменная x должна была изменить своё значение на 2. Вы можете это проверить и попрактиковаться на примере в виде файла!

Окно просмотра

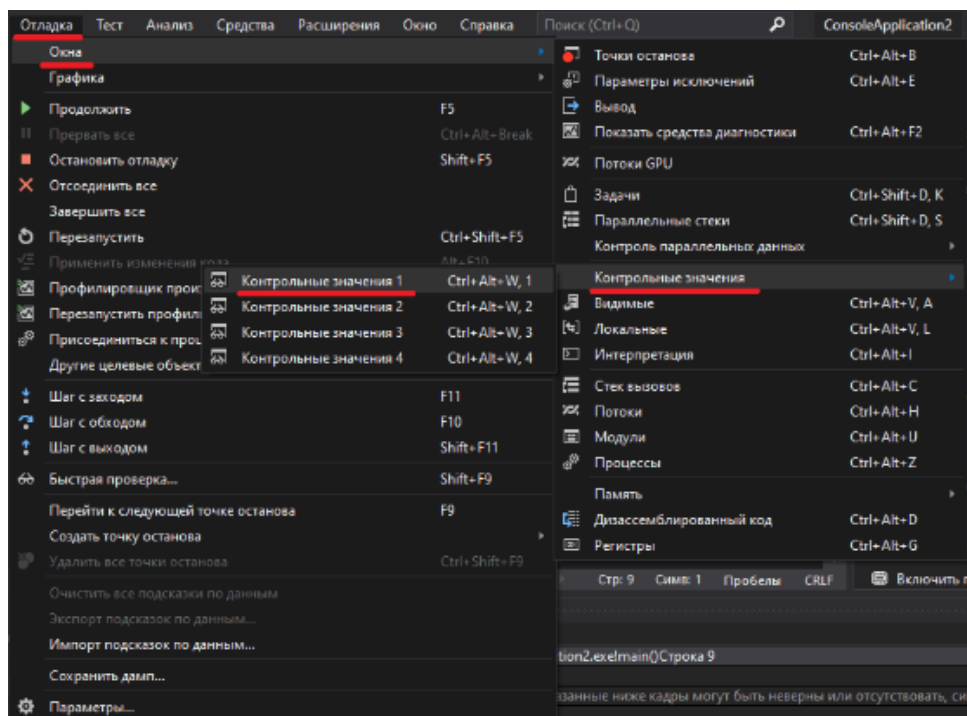
Если вы уже знаете, как пользоваться окном просмотра, переходите к следующей под-главе.

«Быстрая проверка» или наведения курсора – не единственные способы посмотреть значения переменных. Также они не являются удобными или эффективными, и не очень подходят для отслеживания изменений переменных во время выполнения.

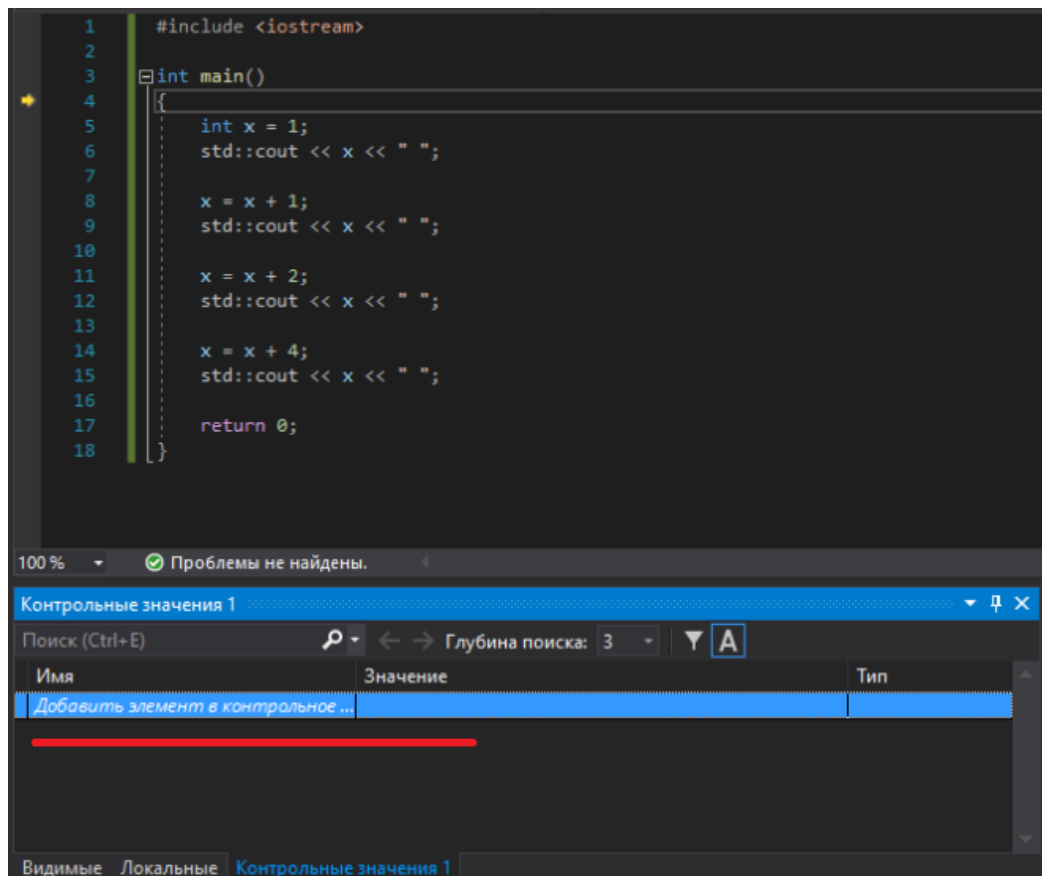
Чтобы не открывать после выполнения каждой команды «Быструю проверку» или не наводить мышь существует инструмент – окно просмотра.

Окно просмотра – окно, в которое можно добавлять переменные для отслеживания. Переменные будут автоматически обновляться по ходу выполнения программы.

Окно просмотра скорее всего уже включено в вашем IDE, но если это не так, то перейдите в пункт меню «Отладка» => «Окна» => «Контрольные значения» => «Контрольные значения 1»:



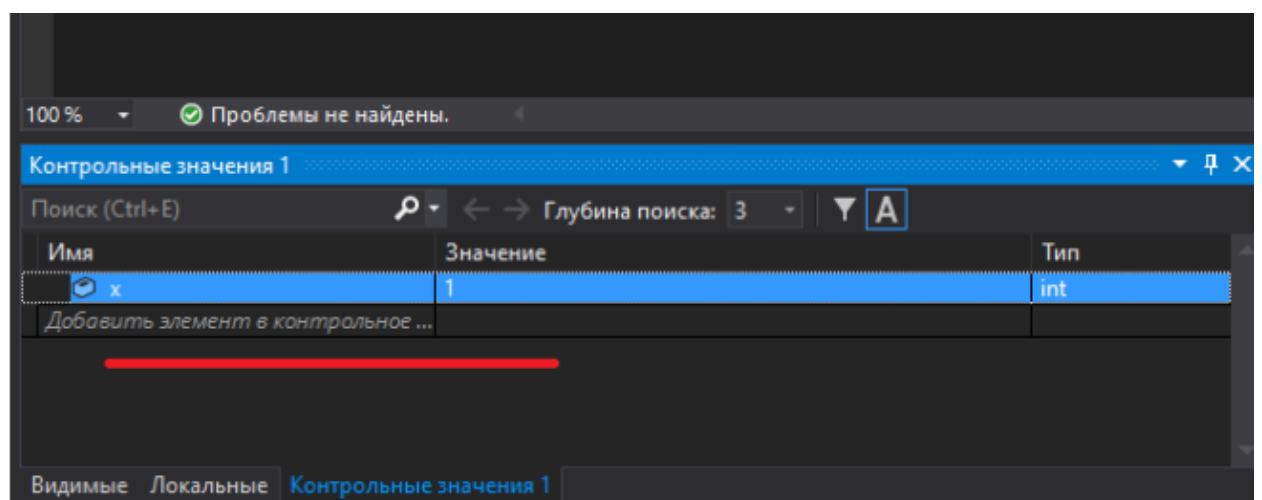
После того, как вы это сделаете, вы должны увидеть следующее:



В этом окне пока что ничего нет, так как вы ещё не добавили ничего в отслеживание. Есть два способа это сделать:

- Ввести имя отслеживаемой переменной в колонку «Имя» в окне просмотра
- Выделить переменную и, открыв меню правой кнопкой мыши, нажать «Добавить контрольное значение»

Попробуйте сами добавить переменную x в окно просмотра(используем всё тот же пример):



Теперь вы можете нажимать «Шаг с обходом» и наблюдать за тем, как изменяется ваша переменная!

Стек вызовов

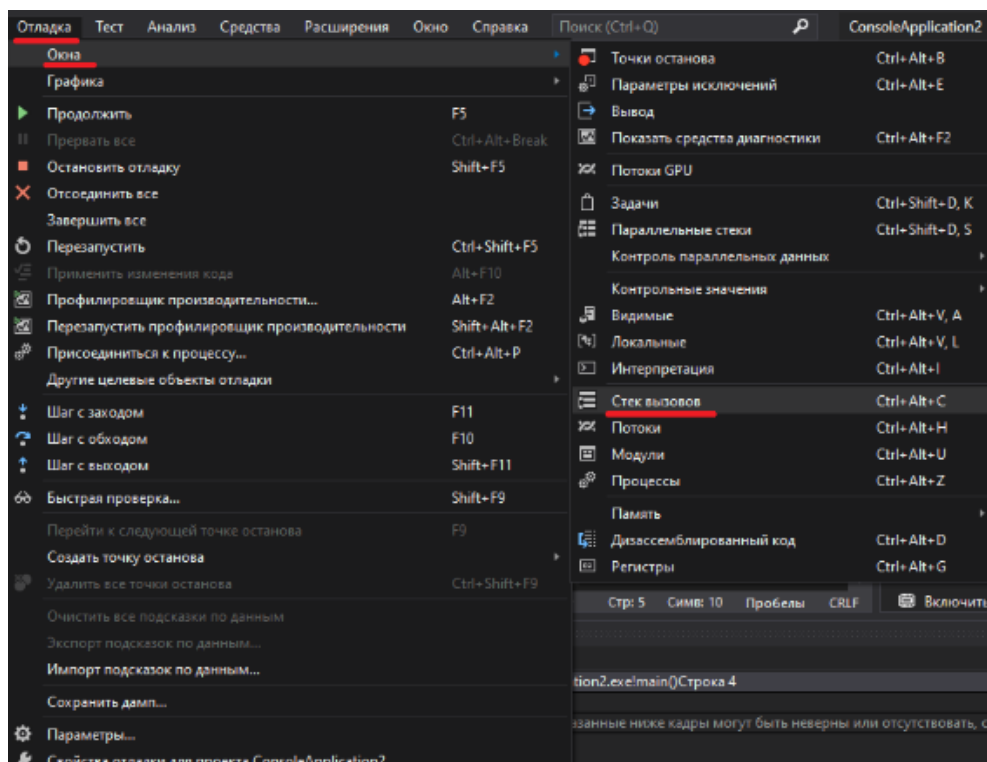
Если вы уже знакомы со стеком вызовов, то переходите к следующей подглаве.

Современные отладчики имеют еще одно информационное окно, которое может быть очень полезным при отладке программ — «Стек вызовов».

Как вы уже знаете, при вызове функции программа оставляет закладку в текущем местоположении, выполняет функцию, а затем возвращается в место закладки. Программа отслеживает все вызовы функций в стеке вызовов.

Стек вызовов — это список всех активных функций, которые вызывались до текущего местоположения. В стек вызовов записывается вызываемая функция и выполняемая строка. Всякий раз, когда происходит вызов новой функции, эта новая функция добавляется в верх стека. Когда выполнение текущей функции прекращается, она удаляется из верхней части стека и управление переходит к функции ниже (второй по счету).

Отобразить окно «Стека вызовов» в Visual Studio можно через "Отладка" => "Окна" => "Стек вызовов":



Примечание: вы должны находиться в режиме отладки

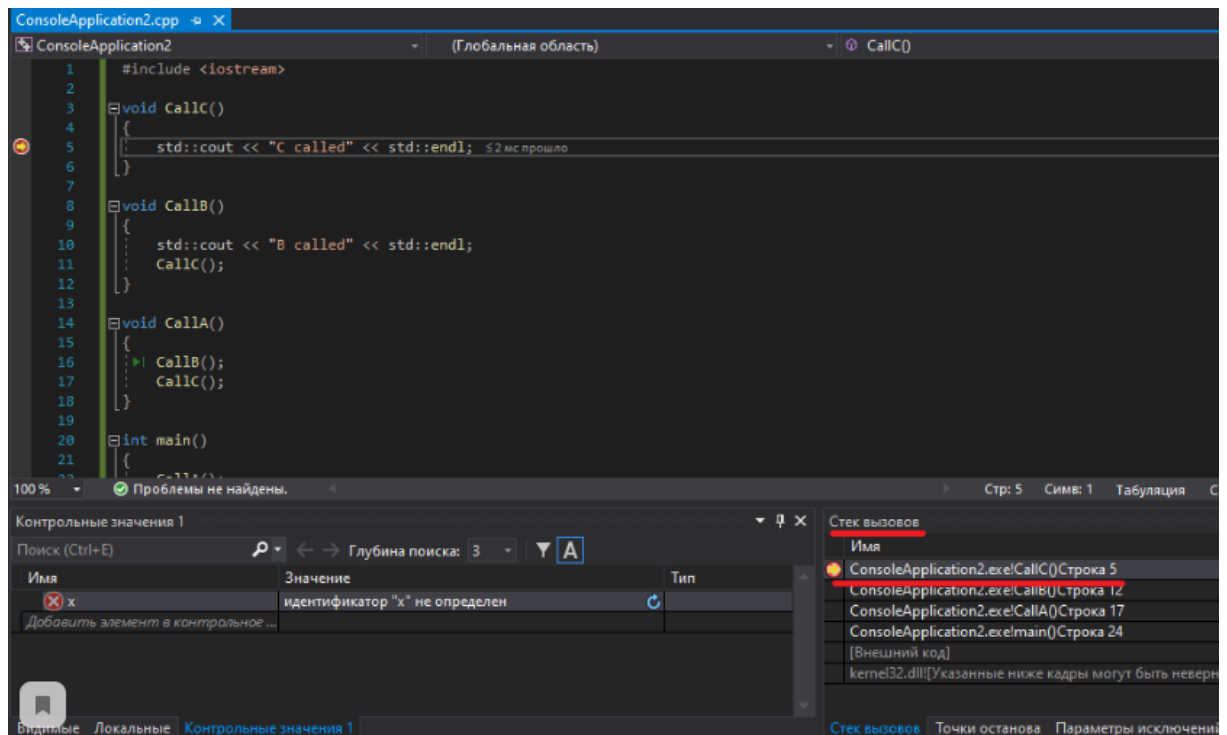
Рассмотрим третий пример. Исходный код находится тут:
«<https://github.com/12PAIN/UchPrakt/tree/main/Examples/C%2B%2B>»
Название файла: Example_3.cpp.

Исходный код:

```
1 #include <iostream>
2
3 void CallC()
4 {
5     std::cout << "C called" << std::endl;
6 }
7
8 void CallB()
9 {
10    std::cout << "B called" << std::endl;
11    CallC();
12 }
13
14 void CallA()
15 {
16    CallB();
17    CallC();
18 }
19
20 int main()
21 {
22    CallA();
23
24    return 0;
25 }
```

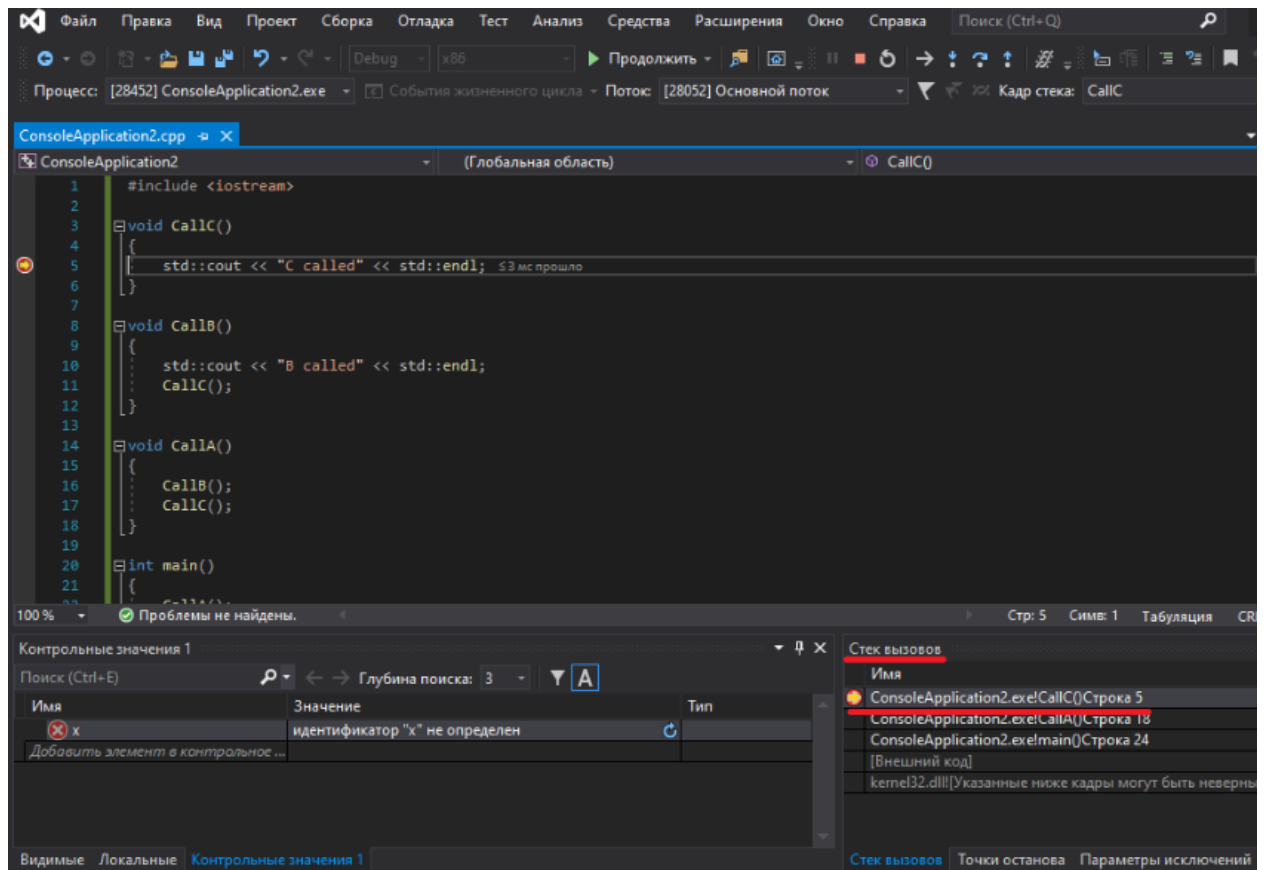
Укажите точку останова в функции CallC(), а затем запустите отладку. Программа выполнится до точки останова.

Несмотря на то, что вы знаете, что сейчас выполняется CallC(), в программе есть два вызова CallC(): в функции CallB() и в функции CallA(). Какая функция ответственна за вызов CallC() в данный момент? Стек вызовов нам это покажет:



Сначала выполняется `main()`. Затем `main()` вызывает `CallA()`, которая, в свою очередь, вызывает `CallB()`. Функция `CallB()` вызывает `CallC()`. Вы можете щелкнуть дважды по разным строкам в окне «Стек вызовов», чтобы увидеть больше информации о вызываемых функциях. Некоторые IDE переносят курсор непосредственно к вызову указанной функции. Visual Studio переносит курсор к следующей строке, которая находится после вызова функции. Попробуйте! Для того, чтобы возобновить степпинг, щелкните дважды по самой верхней (первой) строке в окне «Стек вызовов» и вы вернетесь к текущей точке выполнения.

Выберите команду «Продолжить». Точка останова должна сработать во второй раз, когда будет повторный вызов функции `CallC()` (на этот раз из функции `CallA()`). Всё происходящее вы должны увидеть в окне «Стек вызовов»:



Теперь вы знаете об основных возможностях встроенных отладчиков! Используя степпинг, точки останова, отслеживание переменных и окно «Стек вызовов» вы можете успешно проводить отладку программ.

Действия точки останова и точки трассировки.

Условия точки останова

Если вы уже знакомы с условиями точек останова и точками трассировки, то переходите к следующей подглаве.

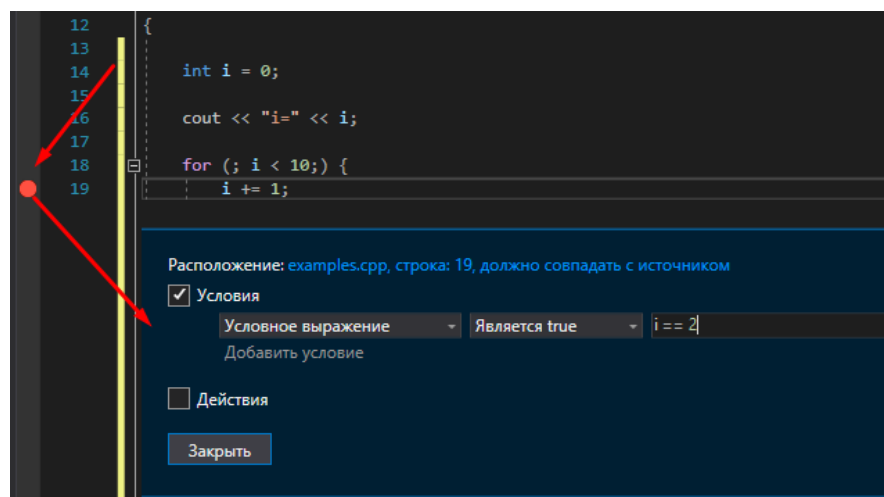
Точка трассировки — это точка останова, которая выводит сообщение в окно вывода. Точка трассировки может играть роль временного оператора трассировки в языке программирования и не приостанавливает выполнение кода. Вы создаете точку трассировки, задавая особое действие в окне Параметры точки останова

- **Условия точки останова**

Можно управлять тем, где и когда выполняется точка останова, задавая условия. Условие может быть любым допустимым выражением, которое распознает отладчик

Задание условия для точки останова:

1. Щелкните правой кнопкой мыши символ точки останова и выберите пункт Условия (или нажмите клавиши ALT + F9, C). Или наведите курсор на символ точки останова, выберите значок Параметры, а затем выберите Условия в окне Параметры точки останова. Можно также задать условия в окне Точки останова, щелкнув правой кнопкой мыши точку останова и выбрав пункт Параметры, а затем Условия



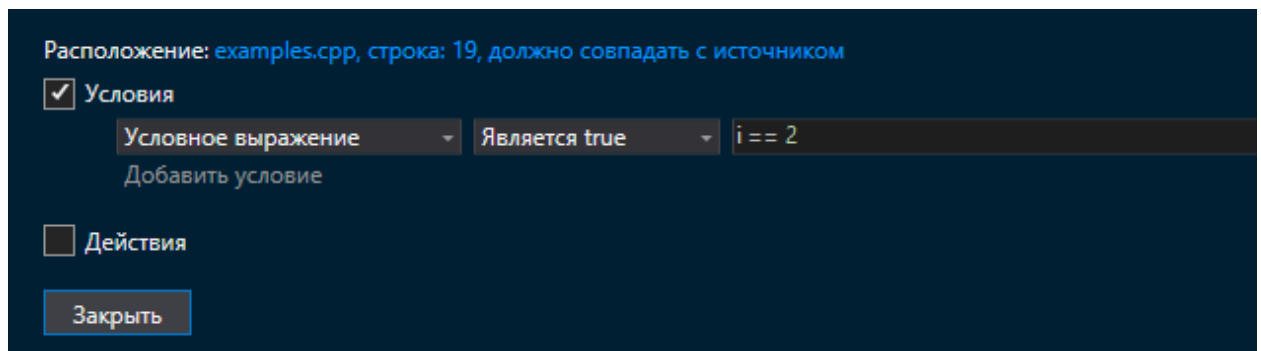
2. В раскрывающемся списке выберите Условное выражение, Количество обращений или Фильтр и задайте соответствующее значение

3. Выберите Закрывать или нажмите клавиши CTRL+BВВОД, чтобы закрыть окно Параметры точки останова. Или в окне Точки останова выберите ОК, чтобы закрыть диалоговое окно

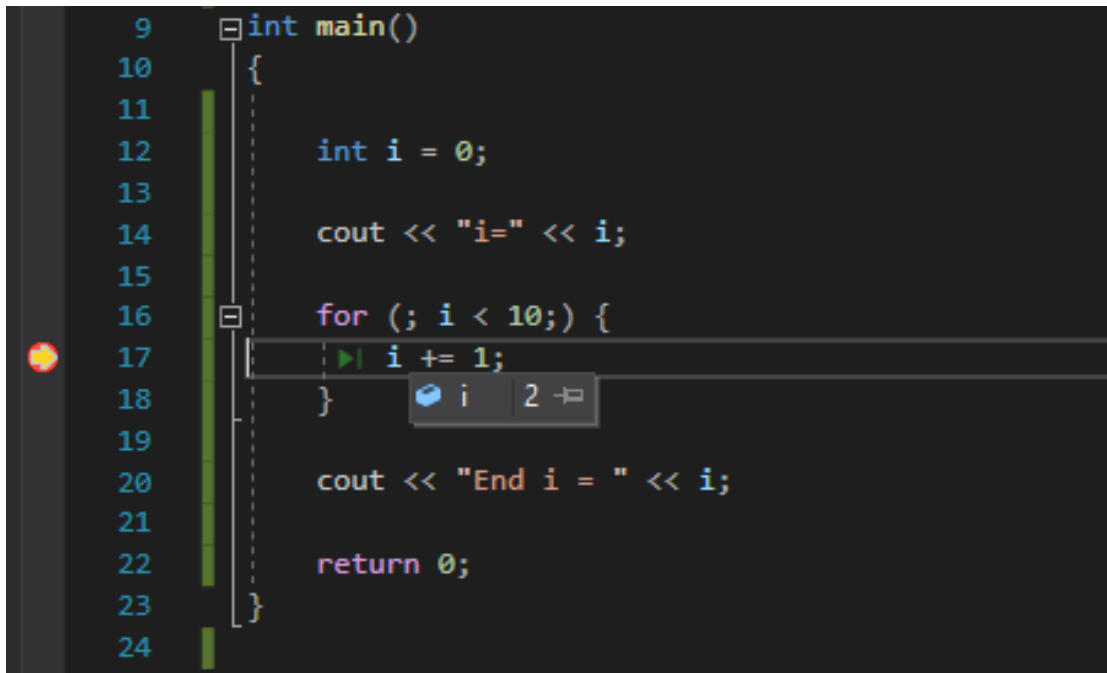
- **Условные выражения:**

Если вы выбрали Условное выражение, можно выбрать одно из двух условий: Имеет значение true или При изменении. Выберите значение Имеет значение true, чтобы прервать выполнение при истинности выражения, или значение При изменении, если требуется прервать выполнение при изменении значения выражения

В примере вы видите автоматическую переменную I. Давайте попробуем задать условие остановки, когда значение I будет равно 2. Для этого создаём условное выражение в точке останова такого вида: `i == 2`



Закроем и попробуем выполнить программу:



```
9  int main()
10 {
11
12     int i = 0;
13
14     cout << "i=" << i;
15
16     for (; i < 10;) {
17         i += 1;
18     }
19
20     cout << "End i = " << i;
21
22     return 0;
23 }
24
```

The screenshot shows a C++ IDE with a dark theme. A red breakpoint icon is set on line 17, which is `i += 1;`. A tooltip for the variable `i` is visible, showing its current value as 2. The code is a simple program that initializes `i` to 0, prints its value, enters a loop that increments `i` until it reaches 10, prints the final value, and returns 0.

Как вы можете увидеть – значение переменной `i` равняется двум, а программа приостановлена

Вы можете сами попробовать это сделать на этом же примере:

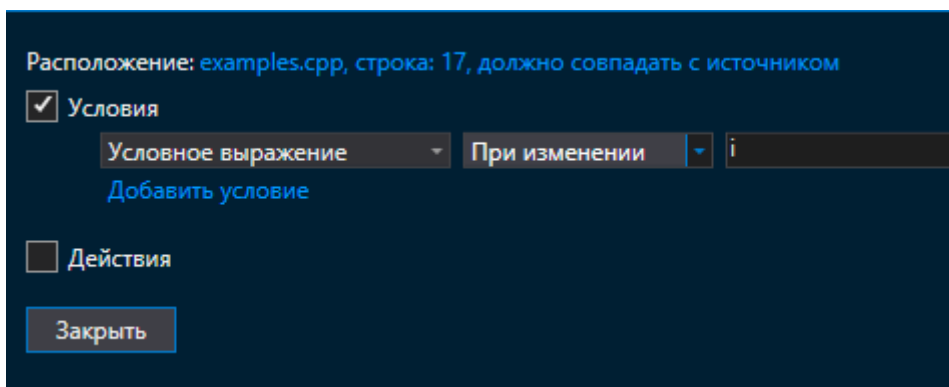
Исходный код находится тут:

«<https://github.com/12PAIN/UchPrakt/tree/main/Examples/C%2B%2B>»

Название файла: `Example_4.cpp`

Также можно задать условное выражение у точки останова, такое, что программа будет останавливаться при любом изменении значения переменной

Давайте попробуем это сделать в этом же примере:



Расположение: `examples.cpp`, строка: 17, должно совпадать с источником

☒ Условия

Условное выражение: При изменении `i`

[Добавить условие](#)

☐ Действия

Запустим:

```
9  int main()
10 {
11
12     int i = 0;
13
14     cout << "i=" << i;
15
16     for (; i < 10;) {
17         i += 1;
18     }
19     cout << "End i = " << i;
20
21     return 0;
22 }
23
24
```

```
9  int main()
10 {
11
12     int i = 0;
13
14     cout << "i=" << i;
15
16     for (; i < 10;) {
17         i += 1; ≤ 1 мс прошло
18     }
19     cout << "End i = " << i;
20
21     return 0;
22 }
23
24
```

Как вы видите, программа приостанавливается, если изменяется значение *i*. Также вы сами можете попробовать это сделать на этом же самом примере

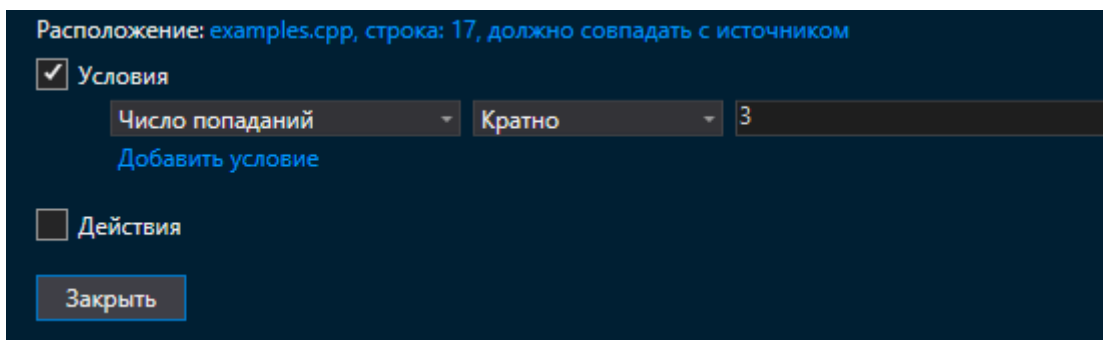
Учтите, что если условие точки останова имеет недопустимый синтаксис, появится предупреждающее сообщение. Если указать условие для точки останова с недопустимой семантикой, но допустимым синтаксисом, предупреждающее сообщение появится при достижении точки останова в первый раз. В любом случае отладчик прерывает выполнение при попадании на недопустимую точку останова. Точка останова пропускается, только если условие допустимо и принимает значение «false».

- **Условное выражение: число попаданий**

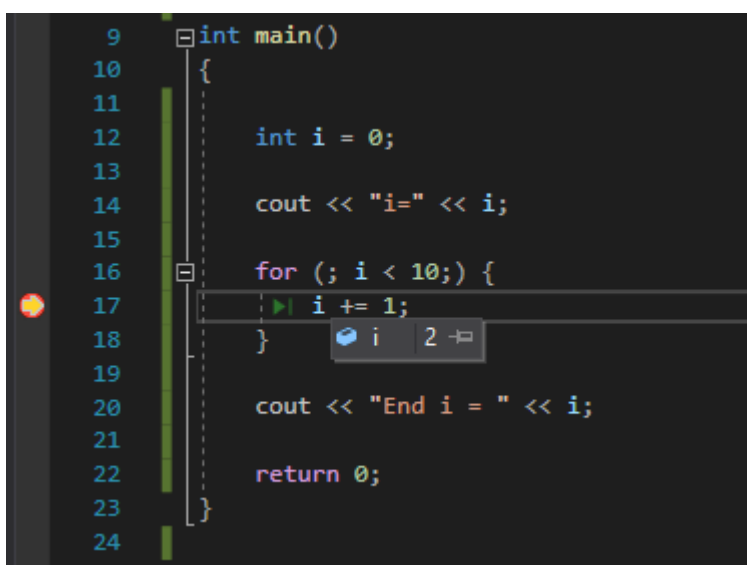
Если есть подозрение, что цикл в коде начинает неправильно вести себя после определенного числа итераций, можно установить

точку останова для остановки выполнения после указанного количества обращений, вместо того чтобы многократно нажимать клавишу F5 для достижения этой итерации.

В разделе Условия в окне Параметры точки останова выберите Количество обращений, а затем укажите число итераций. В примере (использован Example_4.cpp) задается выполнение точки останова при каждой третьей итерации:



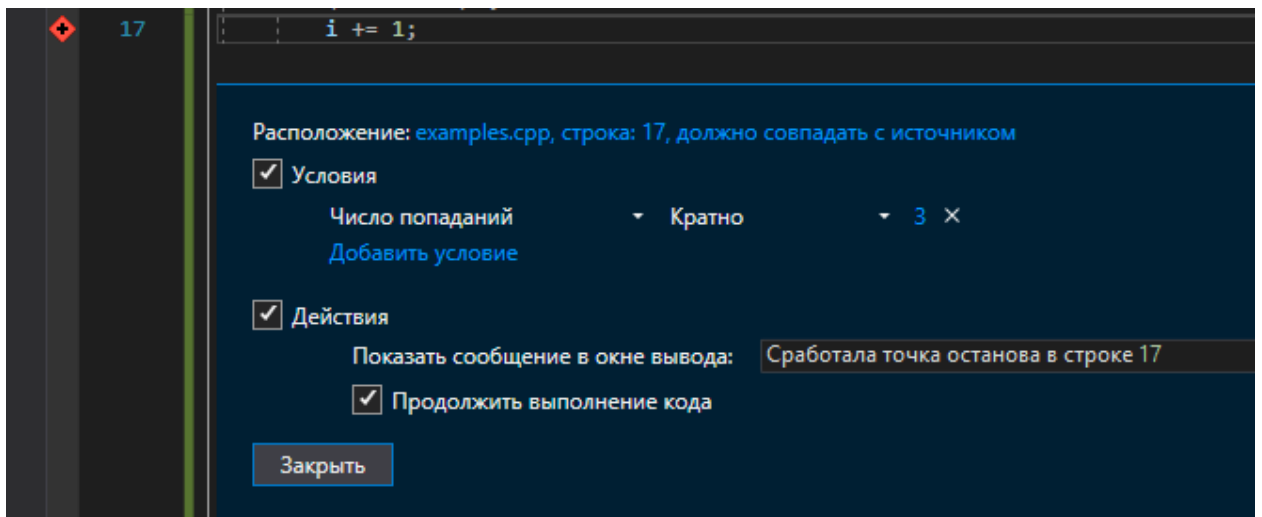
Давайте попробуем запустить программу:



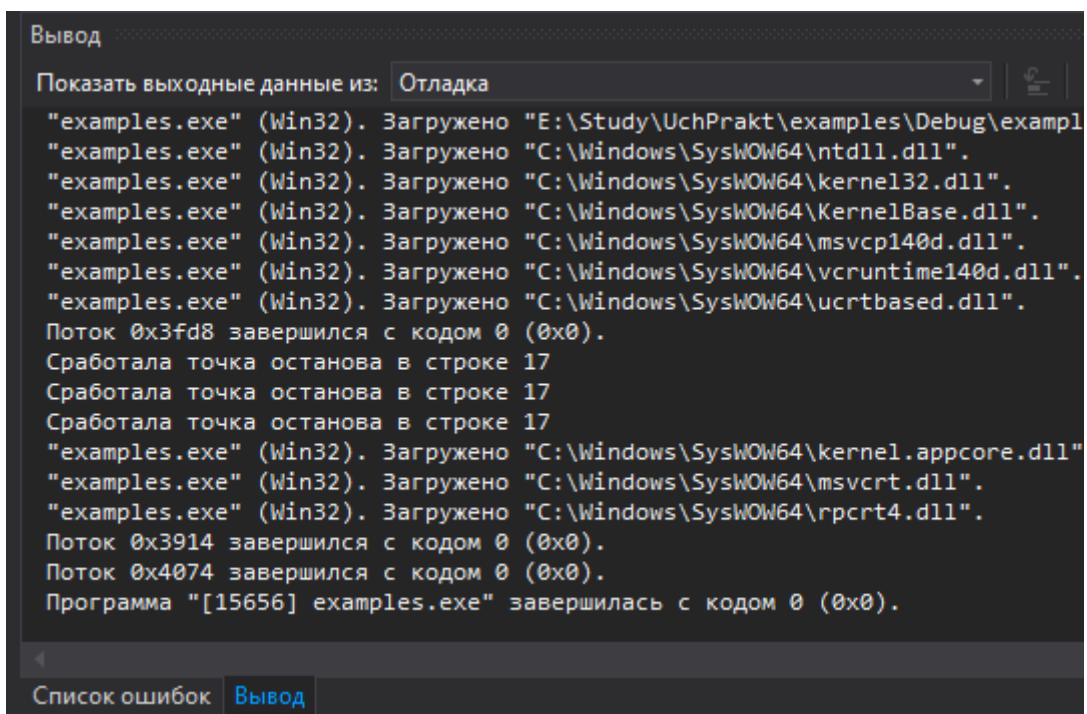
Заметим, что $I = 2$ не потому, что отладчик посчитал, что 2 кратно 3, а потому что изначально у нас $I = 0$, а когда I становится равным 2, то это третье попадание в цикл. Попробуйте сами провести те же самые действия на этом же примере!

- **Действия точки останова**

Можно задать действие для точки останова. То есть – вывод в панель вывода. Пример(всё тот же пример Example_4.cpp):



Мы задали вывод такой вот надписи в строку вывода, при этом мы не останавливаем программу. Флаг «Продолжить выполнение кода» может быть активирован, тогда точка останова не остановит программу, а просто выполнит действие, указанное в пункте «Показать сообщение в окне вывода». Давайте попробуем запустить программу и посмотреть, что же происходит в окне вывода нашей среды программирования (не в окне самой программы):



Как вы можете заметить, у нас вывелись три раза строки, которые мы написали в действии точки останова.

Задание точки останова функции

Если вы уже знакомы с заданием точек останова функций, то переходите к следующей подглаве

Выполнение можно прерывать при вызове функции. Это полезно, например, если известно имя функции, но не ее расположение. Это также полезно, если у вас есть функции с одинаковым именем и вы хотите приостановить их все (например, перегруженные функции или функции в разных проектах).

Установка точки останова функции:

1. Выберите Отладка > Создать точку останова > Точка останова функции или нажмите клавиши CTRL + K, В. Можно также выбрать Создать > Точка останова функции в окне Точки останова.

2. В диалоговом окне Новая точка останова функции заполните поле Имя функции. Чтобы уточнить функцию, сделайте следующее:

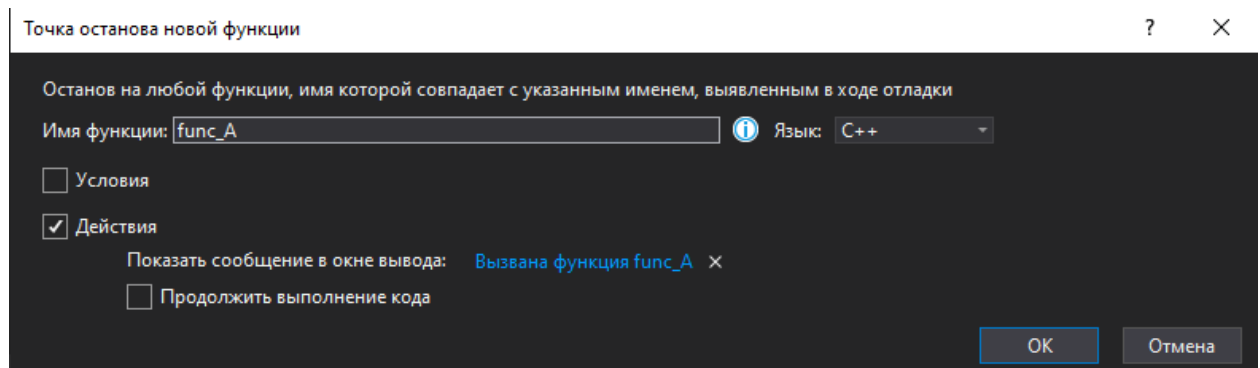
- Используйте полное имя функции.
- Пример: Namespace1.ClassX.MethodA()
- Добавьте типы параметров перегруженной функции.
- Пример: MethodA(int, string)
- Используйте символ "!", чтобы указать модуль.
- Пример: App1.dll!MethodA
- Используйте оператор контекста в машинном коде C++.
- {function, , [module]} [+<line offset from start of method>]
- Пример: {MethodA, , App1.dll}+2

3. В раскрывающемся списке Язык выберите язык функции.

4. Нажмите кнопку ОК.

Пример:

У нас есть функция `func_A`. Давайте зададим точку останова функции на ней:



Запустим программу и проверим. Да, ничего не произойдет, потому что мы не вызываем функцию `func_A`. Давайте перейдем к примеру `Example_5.cpp`

Исходный код находится тут:

«<https://github.com/12PAIN/UchPrakt/tree/main/Examples/C%2B%2B>»

Название файла: `Example_5.cpp`

Создадим такую-же точку останова функции и запустим программу:

The image shows a code editor with a dark theme. The code is as follows:

```
1  #include <iostream>
2
3  using namespace std;
4
5  void func_A() {
6      cout << "Func_A is called";
7  }
8
9  int main()
10 {
11
12     func_A();
13
14     return 0;
15 }
16
```

A red arrow icon, representing a breakpoint, is positioned to the left of line 5, which is the first line of the `func_A` function.

Как вы видите, программа остановилась в функции `func_A`.

Условия и действия точки останова функции могут быть использованы также, как и условия и действия обычной точки останова.

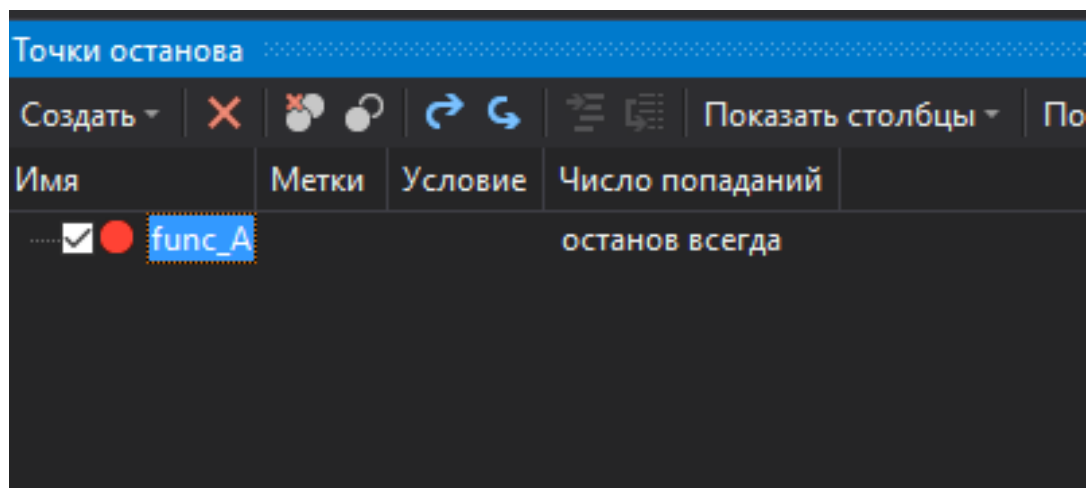
Управление точками останова в окне "Точки останова"

Если вы уже знаете, как управлять точками останова в Visual Studio, то переходите к следующей подглаве

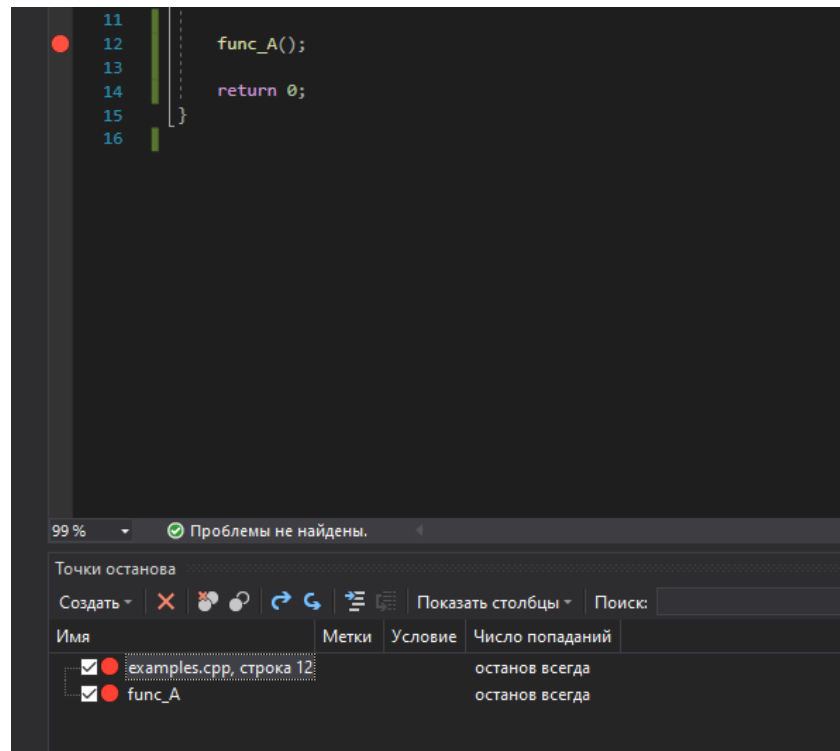
В IDE Visual Studio можно посмотреть все точки останова и управлять ими из отдельного меню, а не в редакторе самой программы.

В окне Точки останова можно выполнять поиск, сортировку, фильтрацию, включение, отключение или удаление точек останова. Можно также задать условия и действия или добавить новую функцию или точку останова в данных.

Чтобы открыть окно Точки останова, выберите Отладка > Windows > Точки останова или нажмите клавиши CTRL+ALT+B.

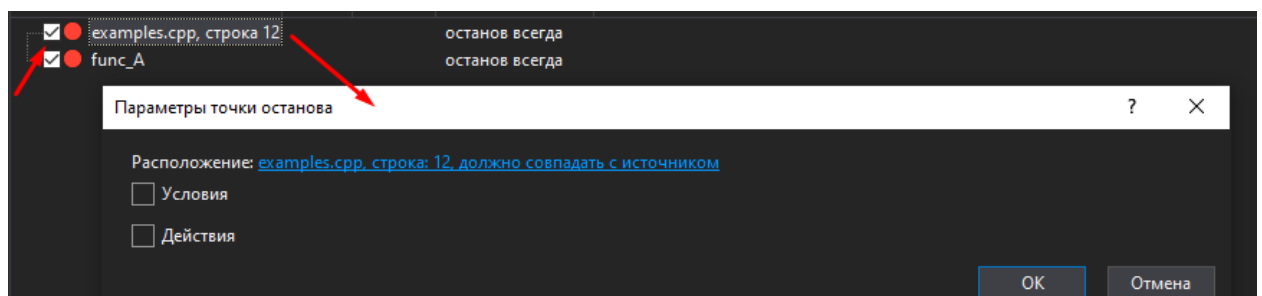


В этом окне вы можете создать новую точку останова в функции. Как вы можете видеть на скриншоте там уже есть точка останова функции func_A. Если мы добавим ещё точку останова, то она отобразится в этом пункте. Давайте сделаем это:



Как вы можете видеть, точка останова сразу отобразилась в окне «Точки останова».

Также в этом окне можно задать действия и условия точек останова. Как это делается вы уже знаете, только здесь нужно нажать правой кнопкой мыши по точке останова и выбрать пункт меню «Параметры»:



Задание точки останова в окне стека вызовов

Чтобы прервать выполнение на инструкции или строке, к которой возвращается вызывающая функция, установите соответствующую точку останова в окне Стек вызовов.

Задание точки останова в окне стека вызовов:

1. Чтобы открыть окно Стек вызовов, необходимо приостановить процесс отладки. Выберите Отладка > Windows > Стек вызовов или нажмите клавиши CTRL+ALT+C.

2. В окне Стек вызовов щелкните правой кнопкой мыши вызывающую функцию и выберите Точка останова > Вставить точку останова или нажмите F9.

3. В левом поле стека вызовов рядом с именем вызова функции появится символ точки останова.

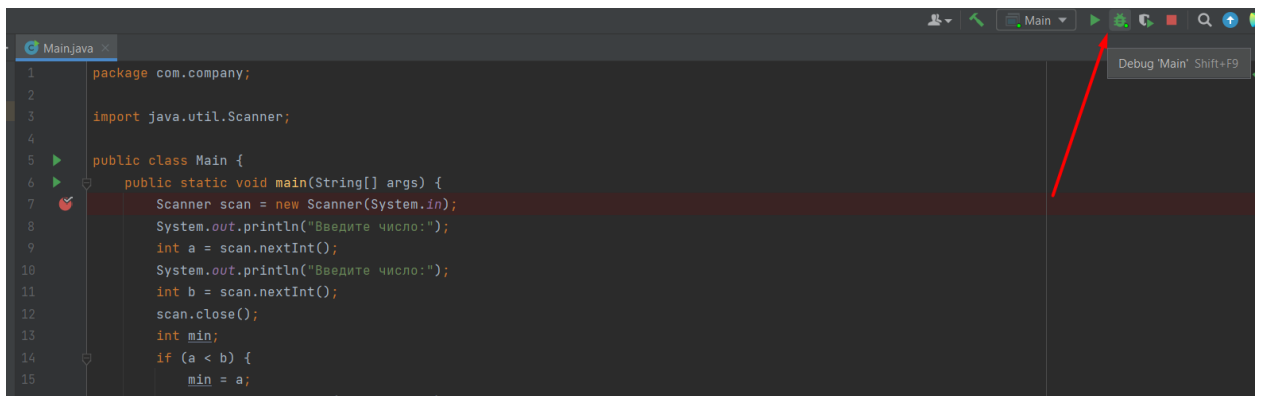
4. В окне Точки останова точка останова стека вызова будет представлена как адрес с областью памяти, который соответствует следующей исполняемой инструкции в функции.

Отладчик приостанавливает выполнение на этой инструкции.

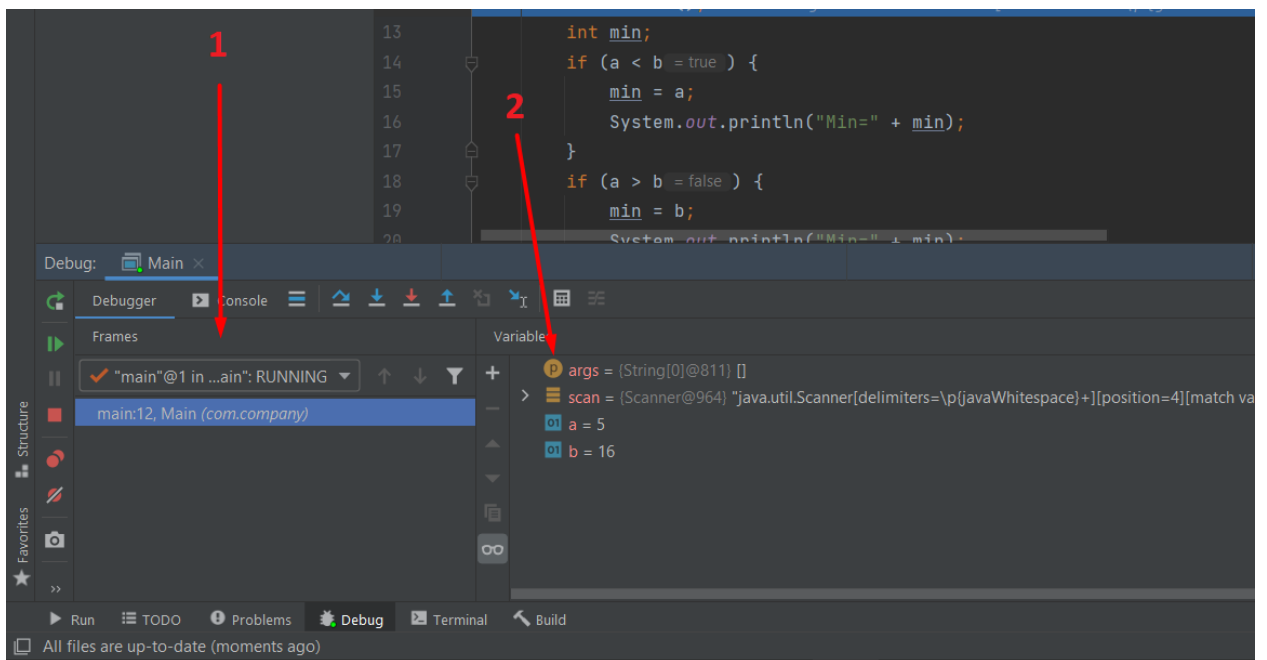
Отладка в IDE IntelliJ IDEA на языке программирования Java

Дебагинг в IntelliJ IDEA

Для того, чтобы начать процесс отладки в данной IDE, нужно нажать сочетание клавиш Shift + F9 или нажать на значок в форме жука в правом верхнем углу, при этом, не забыв поставить точку останова:



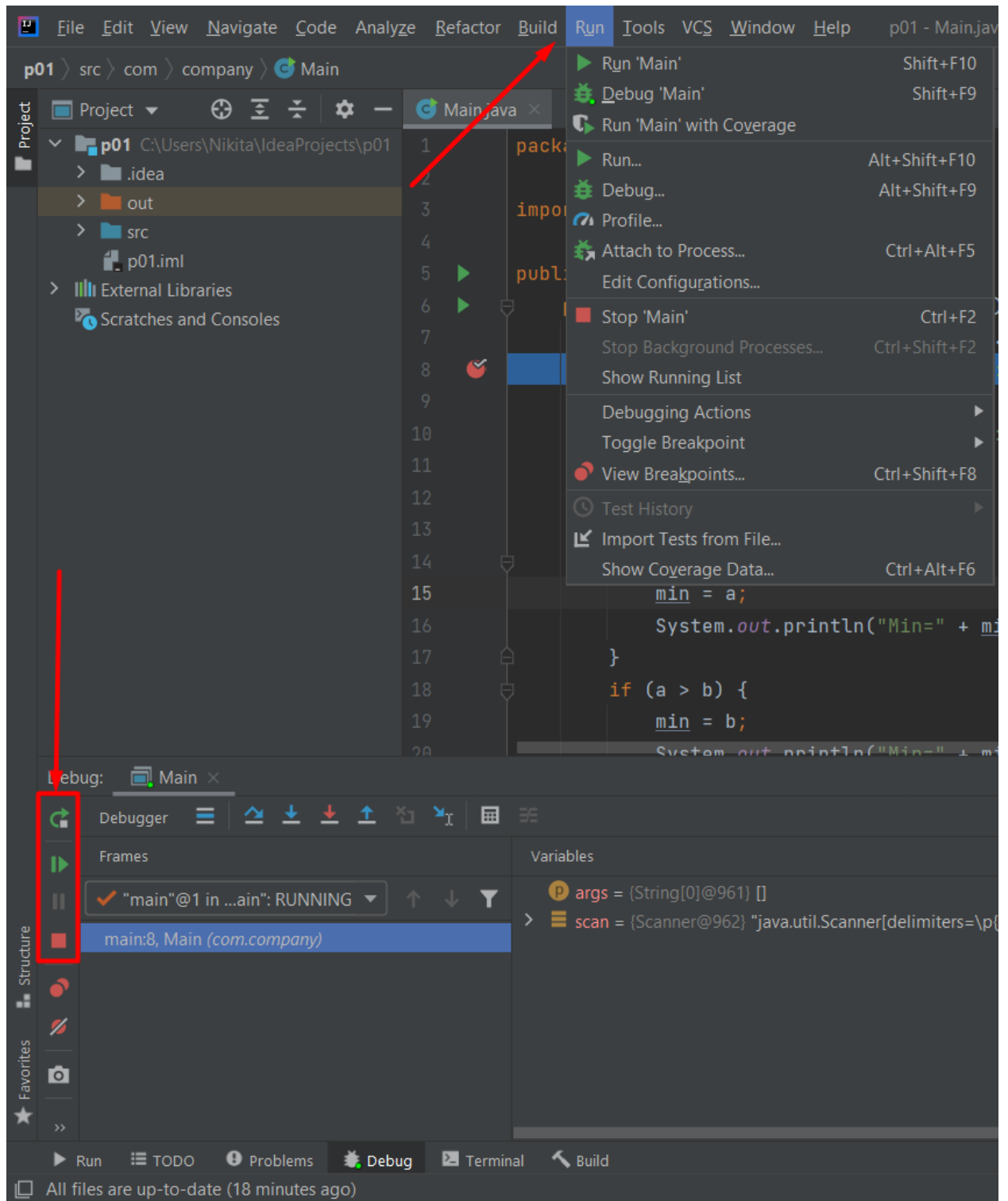
Снизу покажется специальное меню, которое показывает все основные окна для отладки:



Данное окно выводит стек методов

В данном окне будут показываться все переменные данного класса в ходе отладки

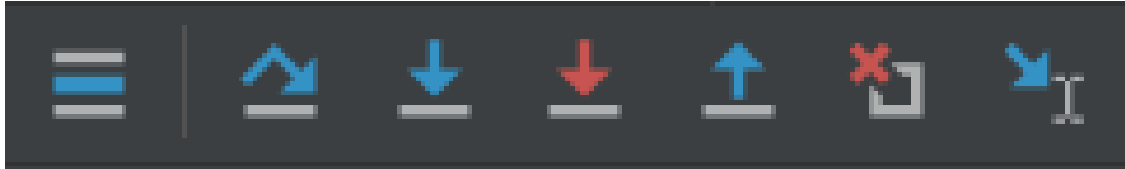
Слева от этих окон будут расположены кнопки для самого процесса отладки либо их можно также найти во вкладке сверху “Run”:



Степпинг

Степпинг (stepping)— это пошаговое выполнение программы.

Когда выполнение кода останавливается на вашей точке, вы можете передвигаться по нему с помощью разных типов шагов. Ниже представлена панель шагов.

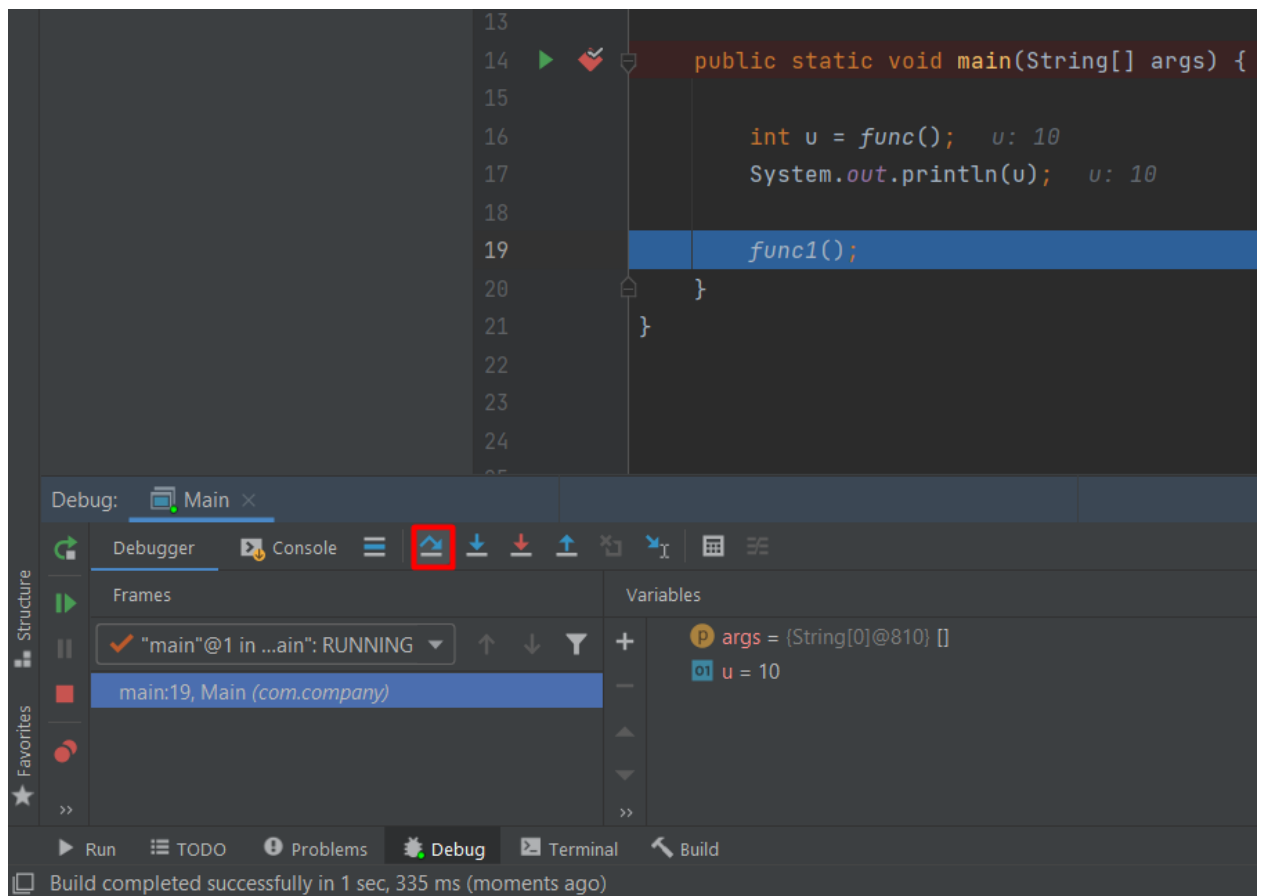


Рассмотрим каждый из них подробнее.

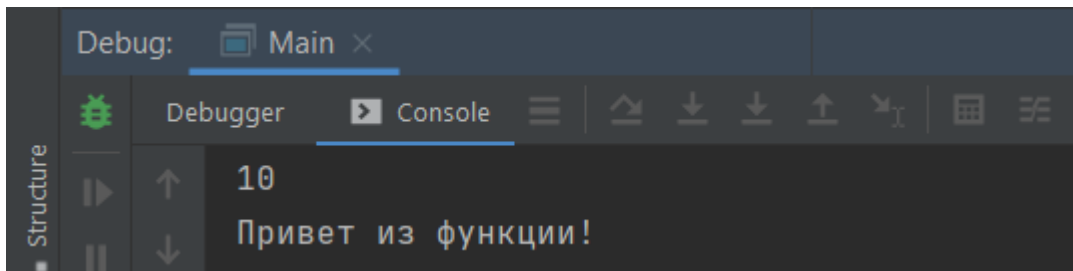
- **Step over (F8)— Шаг с обходом**

Данная команда позволяет перешагнуть через текущую строку кода и перейти к следующей. Реализация методов пропускается, и вы переходите к следующей строке текущего метода

Ставим точку останова на основной блок программы и переходим к отладке. На панели с шагами выбираем “Step over”.



При выборе данного типа шагов программа не будет выполнять заход в функции, которые представлены в коде, а сразу выведет финальное значение.

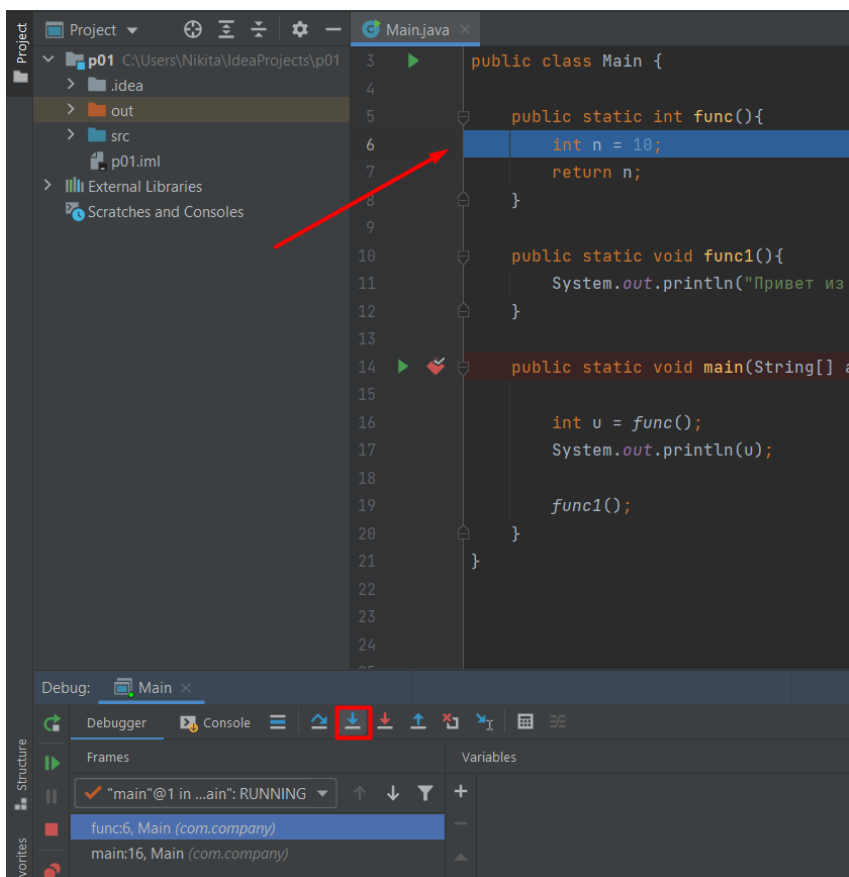


- **Step into (F7)—Шаг с заходом**

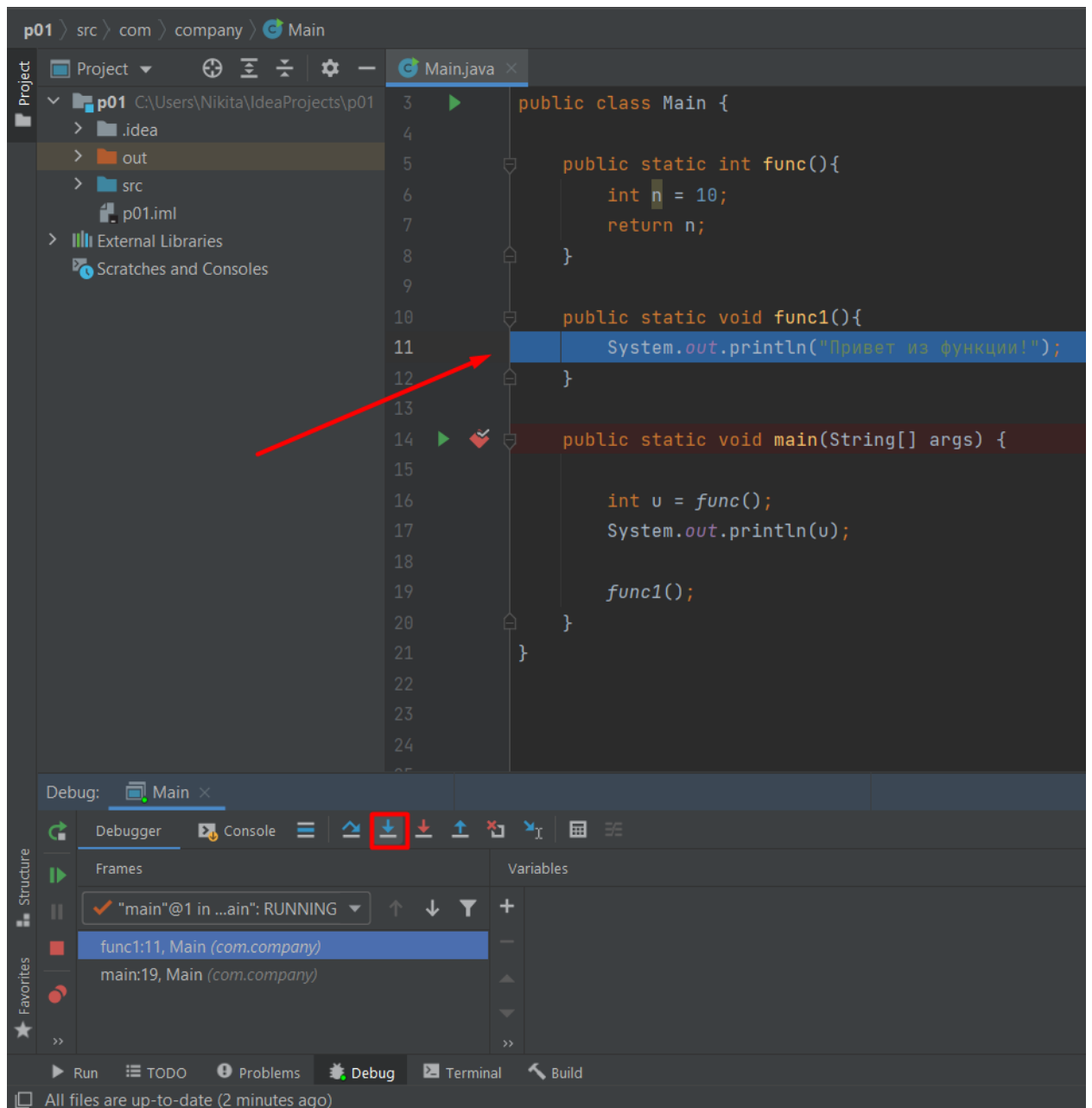
Благодаря этому шагу вы переходите внутрь метода для просмотра его кода. Эта опция подходит для тех случаев, когда вы не уверены, что метод возвращает правильное значение.

Повторяем те же самые действия, что и на прошлом типе шагов, но теперь выбираем кнопку “Step into”. Отличие данного шага от предыдущего в том, что по мере дебагинга, программа будет выполнять заход в функции и построчно идти по ним.

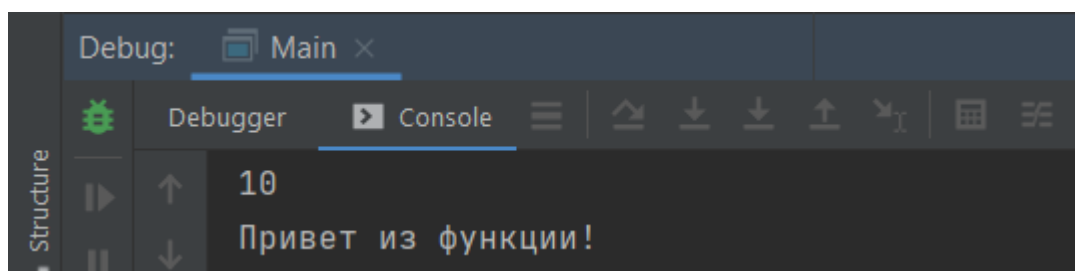
Вход в первую функцию:



Вход во вторую функцию:

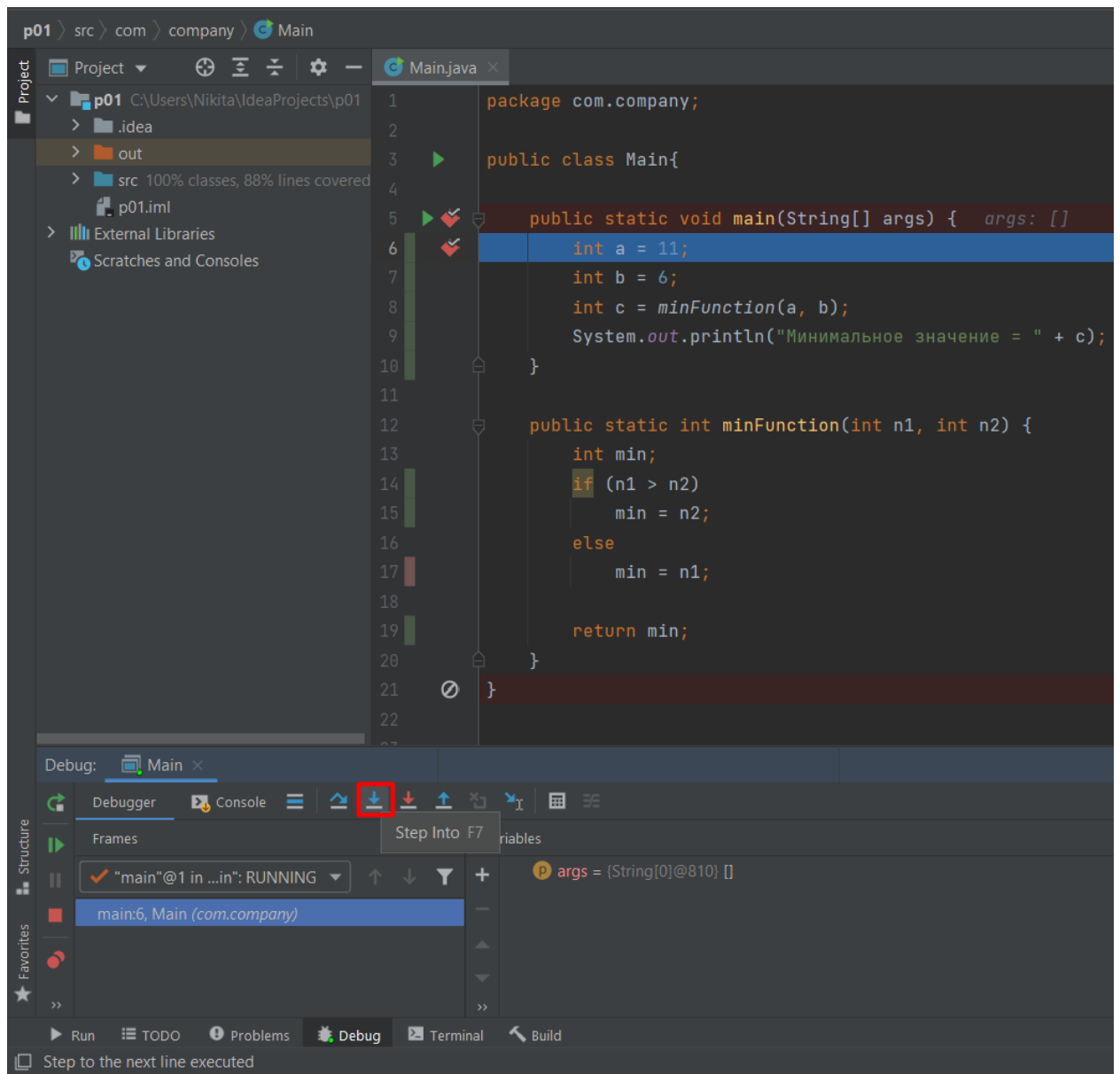


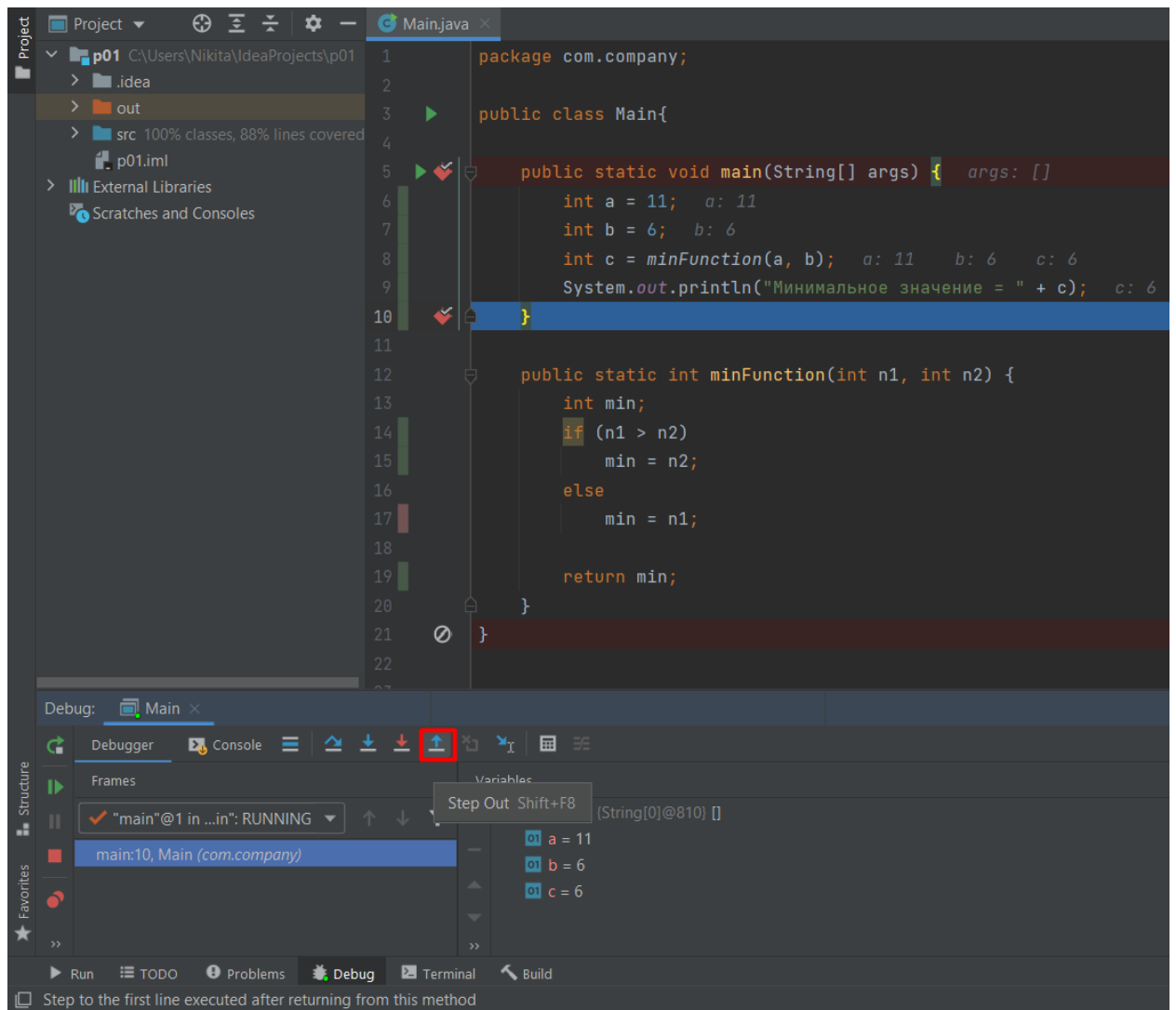
В конечном итоге программа выведет то же самое, что и на предыдущем тип шагов:



- **Step out (Shift + F8)—Шаг с выходом**

При помощи этой команды вы выходите из текущего метода и переходите к вызывающему методу.

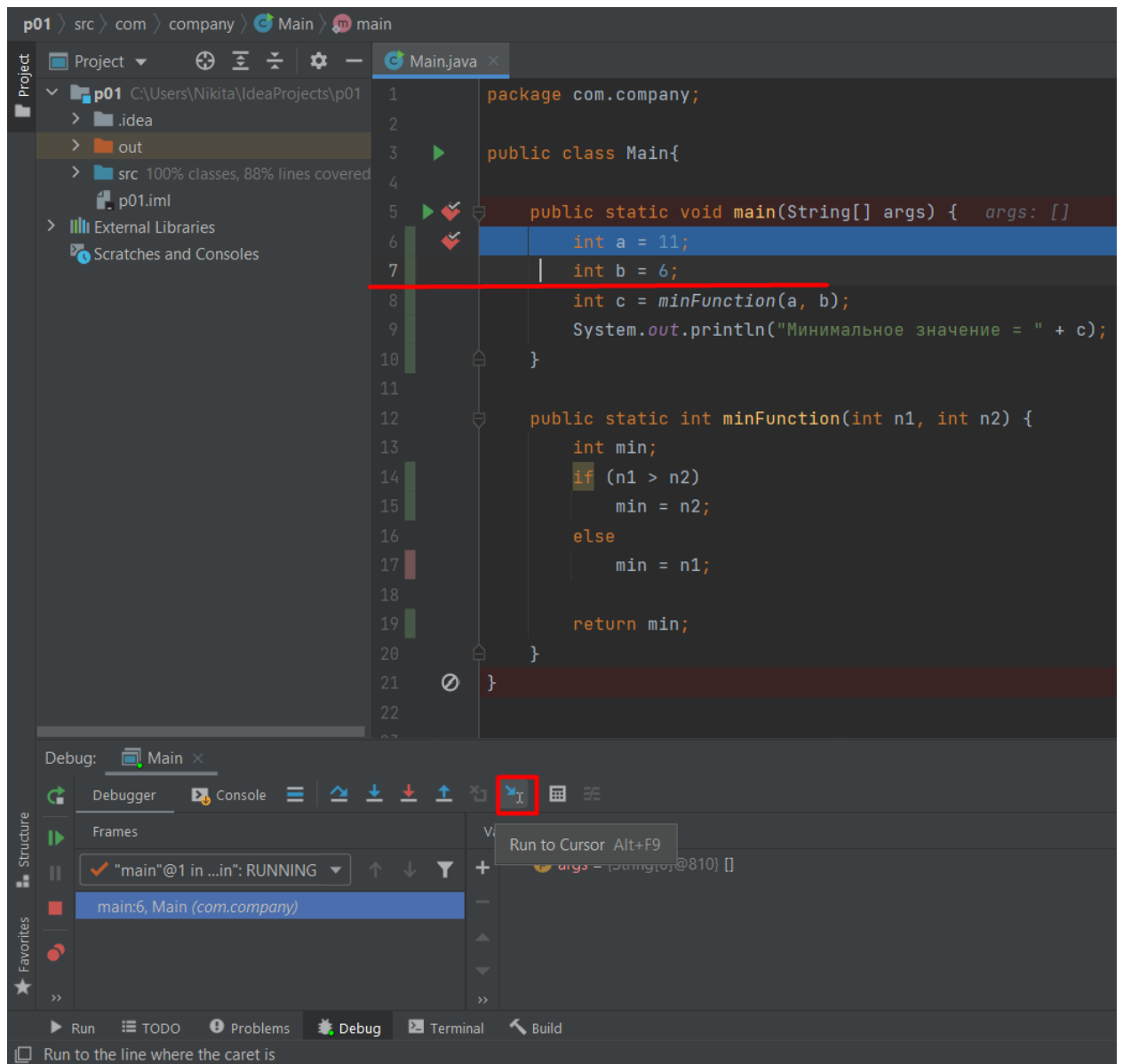




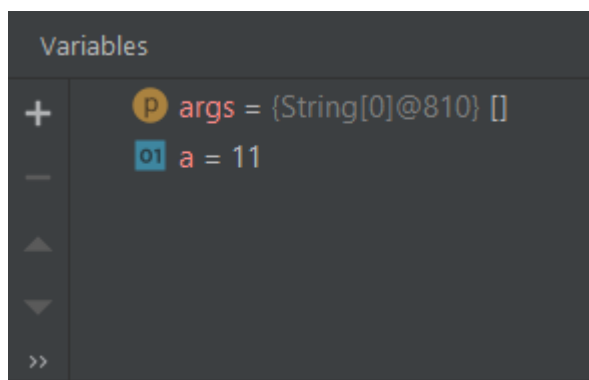
- **Run to cursor(Alt + F9)—Выполнение до курсора**

Данная команда продолжает выполнение приложения до текущей позиции курсора.

Ставим курсор на 7 строчку выбираем “Run to cursor”.



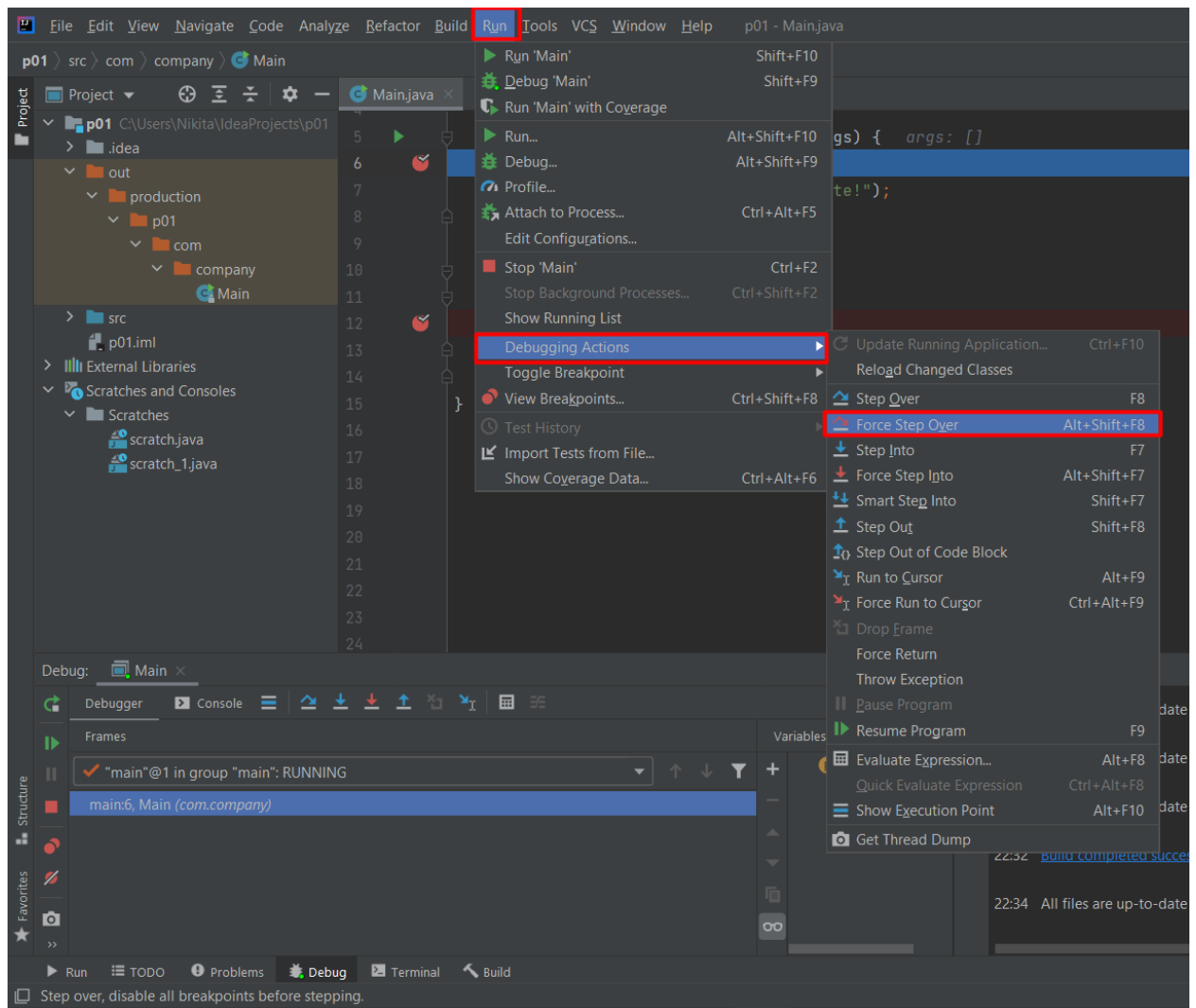
В нижнем окне, которое выводит переменные, можно увидеть, что инициализировалась только переменная *a*. Программа прекратила выполнять свои действия, когда дошла до строки, после которой начинался курсор.



- **Force step over (Shift+Alt+F8)—Принудительный шаг с обходом**

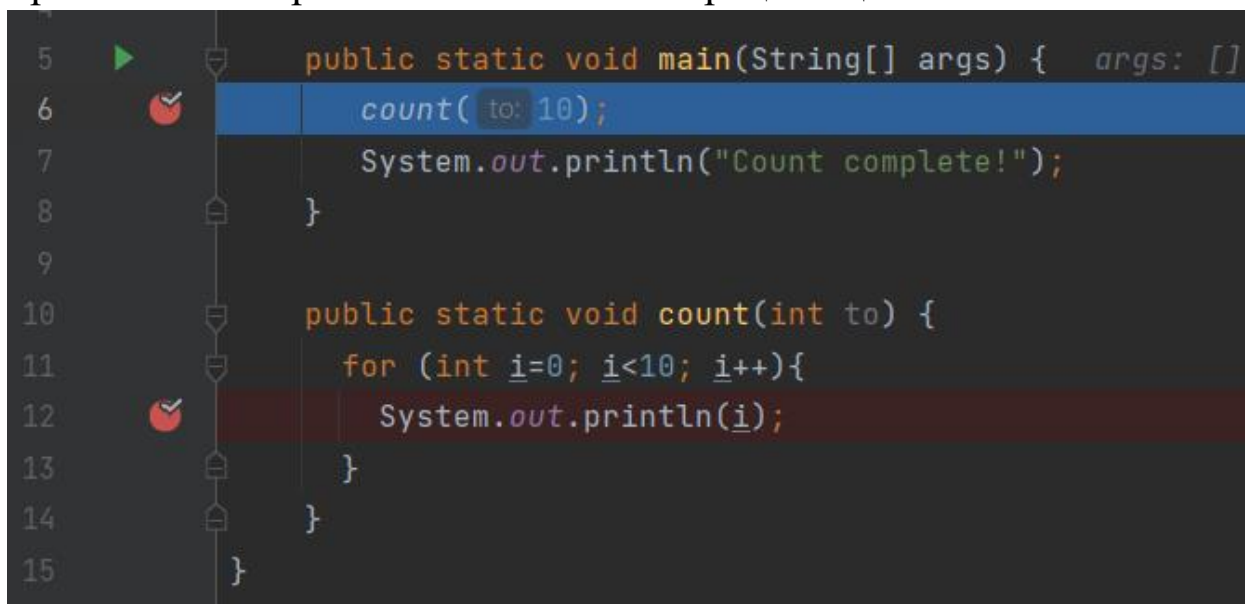
Используя эту команду, вы обходите текущую строку кода и переходите к следующей. Если в вызываемых методах есть точки останова, то они игнорируются.

В главном меню нажмите на вкладку “Run”, затем “Debugging Actions” и “Force Step Over”.



В приведенном примере принудительный переход приведет вас к оператору печати в строке 6, даже если в методе count() есть точка останова, которая в противном случае при переходе через нее

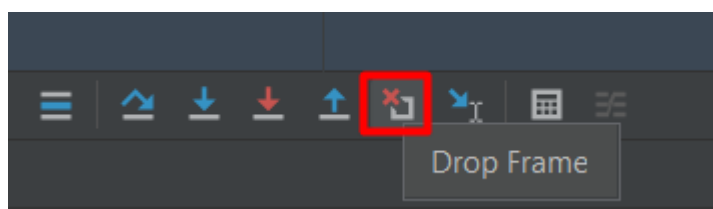
приостановит приложение во всех итерациях цикла.



- **Drop frame—Возврат к предыдущему фрейму**

Эта команда позволяет отменить последний фрейм стека и восстановить предыдущий. Это удобно, например, если вы по ошибке слишком далеко “прошагали” или хотите зайти в функцию, где пропустили важный блок кода.

Обратите внимание, что этот параметр влияет только на локальные переменные и не восстанавливает общее состояние программы, так как не возвращает значения статических переменных и переменных экземпляра. Это может привести к изменению выполнения программы. Чтобы включить “Drop Frame” нужно нажать на соответствующую кнопку:



В этом примере удаление кадра возвращает вас к методу вызывающего объекта, словно count никогда не выполнялся.

```

5  ▶  public static void main(String[] args) {
6      count( to: 10);
7      System.out.println("Count complete!");
8  }
9
10 public static void count(int to) {
11     for (int i=0; i<10; i++){
12         System.out.println(i);
13     }
14 }
15 }

```

Нет статических переменных или переменных экземпляра, которые были затронуты, однако вывод консоли остается.

```

0
1
2

```

Точки останова

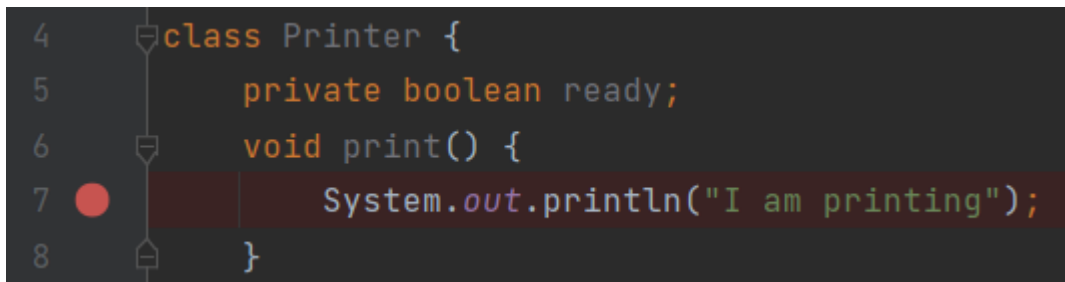
BreakPoint — это специальный маркер, который отображает место или состояние, на котором нужно остановить приложение. Поставить breakpoint можно либо нажав левой кнопкой мыши на левую боковую панель, либо кликнув курсором по месту кода и нажав **Ctrl + F8**.

Breakpoint'ы бывают трех видов:

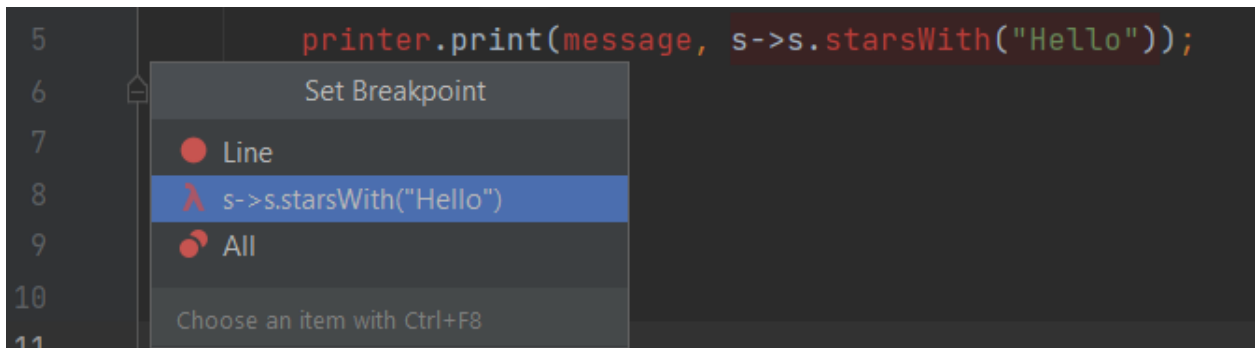
- метка на строку.
- метка на переменную.
- метка на метод.

Выглядит это так:

- На строку:



если в выражении есть лямбда, то IDEA предлагает вам выбор — поставить на всю линию или конкретно в лямбда выражение:



- На метод:

```

3  class Printer {
4      private boolean ready;
5  void print() {
6      System.out.println("I am printing");
7  }
8  }

```

- На класс

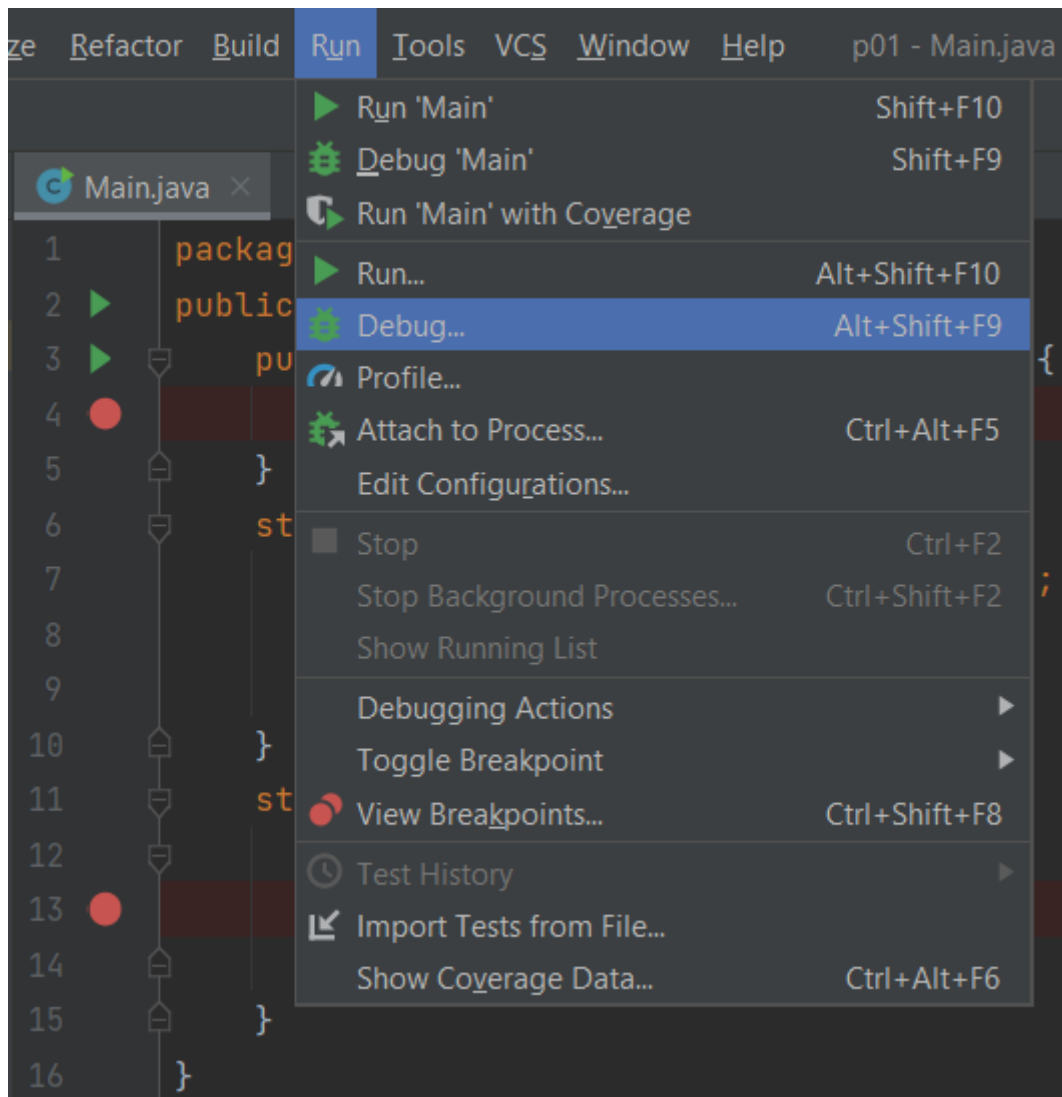
```

3  class Printer {
4      private boolean ready;
5  void print() {
6      System.out.println("I am printing");
7  }
8  }

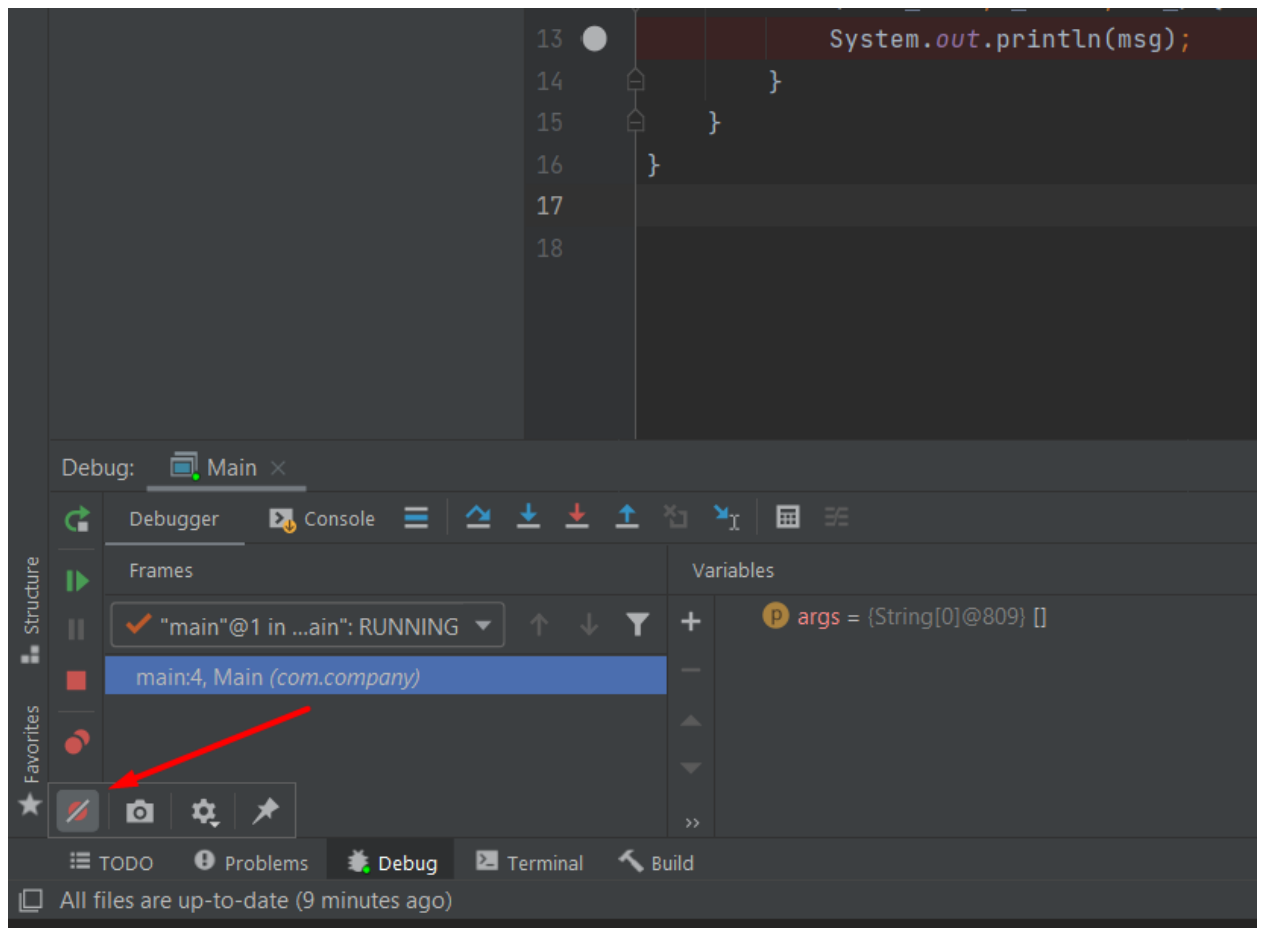
```

Breakpoint'ы можно удалить, путем повторного нажатия левой кнопки мыши на красный кружок.

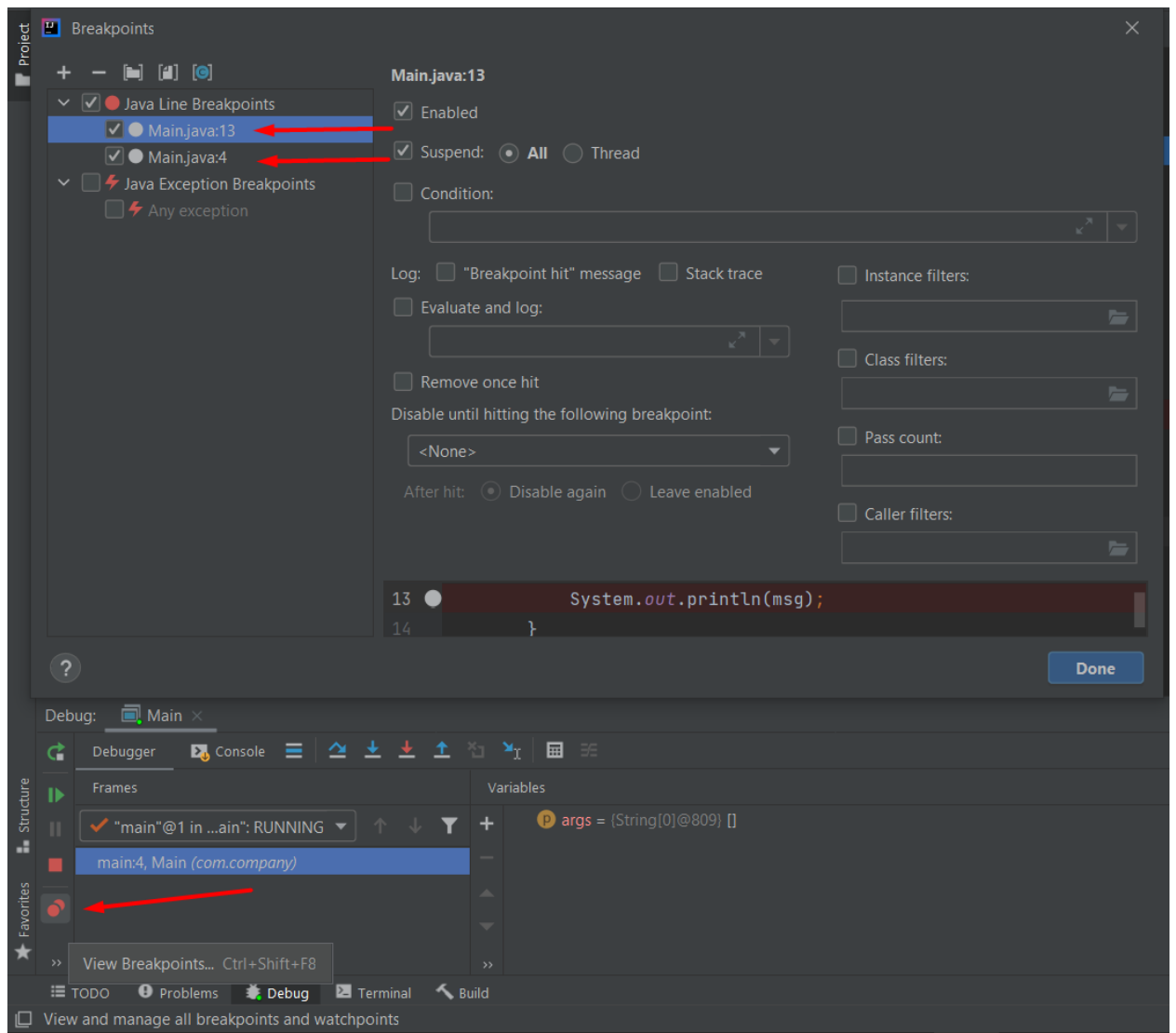
Может возникнуть ситуация, при которой вам понадобится сделать точки останова неактивными. Для этого нужно перейти во вкладку Run в раздел Debug.



Внизу появится консоль для дебагинга, в левом углу которой будет кнопка, чтобы активировать точки останова.



Чтобы посмотреть, какие уже выставленные breakpoint'ы, можно или зайти в Debug в левом нижнем углу, или нажать сочетания клавиш **Ctrl+Shift+F8**. Когда зайдём в список breakpoint'ов, увидим следующее:



Здесь мы можем увидеть, что у нас есть два breakpoint'а:

- Main.java:13 на 13-ой строке
- Main.java:4 на 4-ой строке

Точки останова с условием

Чтобы поставить условие, можно нажать правой кнопкой мыши на breakpoint или перейти в панель просмотра breakpoint'ов

Условием точки останова IntelliJ IDEA может быть любое логическое выражение:

Условие - это логическое выражение Java, включающее метод, возвращающий истину или ложь, например `str1.equals(str2)`.

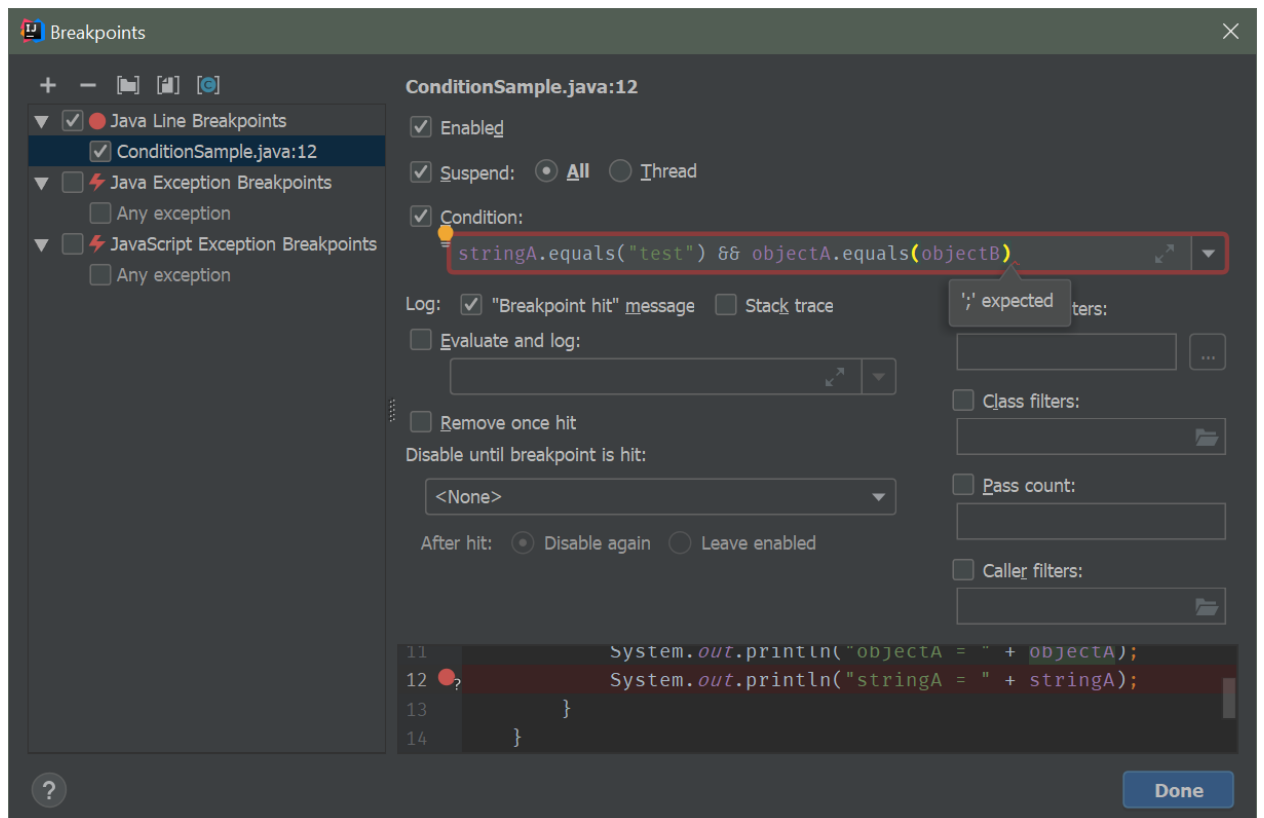
Это выражение должно быть действительным в строке, где установлена точка останова, и оно оценивается каждый раз, когда точка останова достигается. Если результат оценки верен, выбранные действия выполняются.

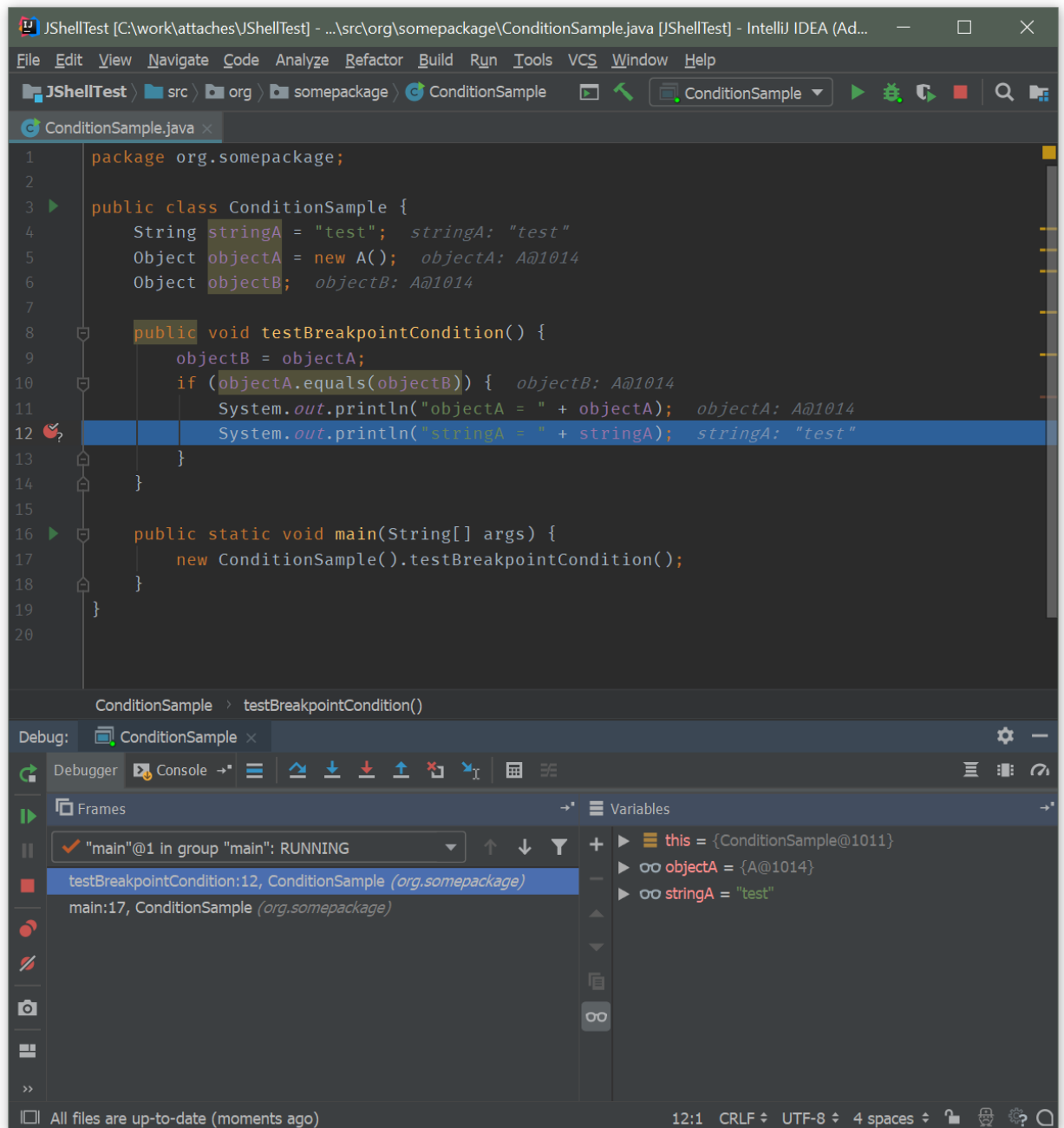
Вы можете вводить многострочные выражения, например:

```
if (myvar == expectedVariable) {  
    System.out.println(myvar);  
    anotherVariable = true;  
}  
return true;
```

`stringA.equals("test") && objectA.equals(objectB)` кажется допустимым выражением, возвращающим `true` или `false`, поэтому оно должно работать сразу после установки.

Доказательство работы:

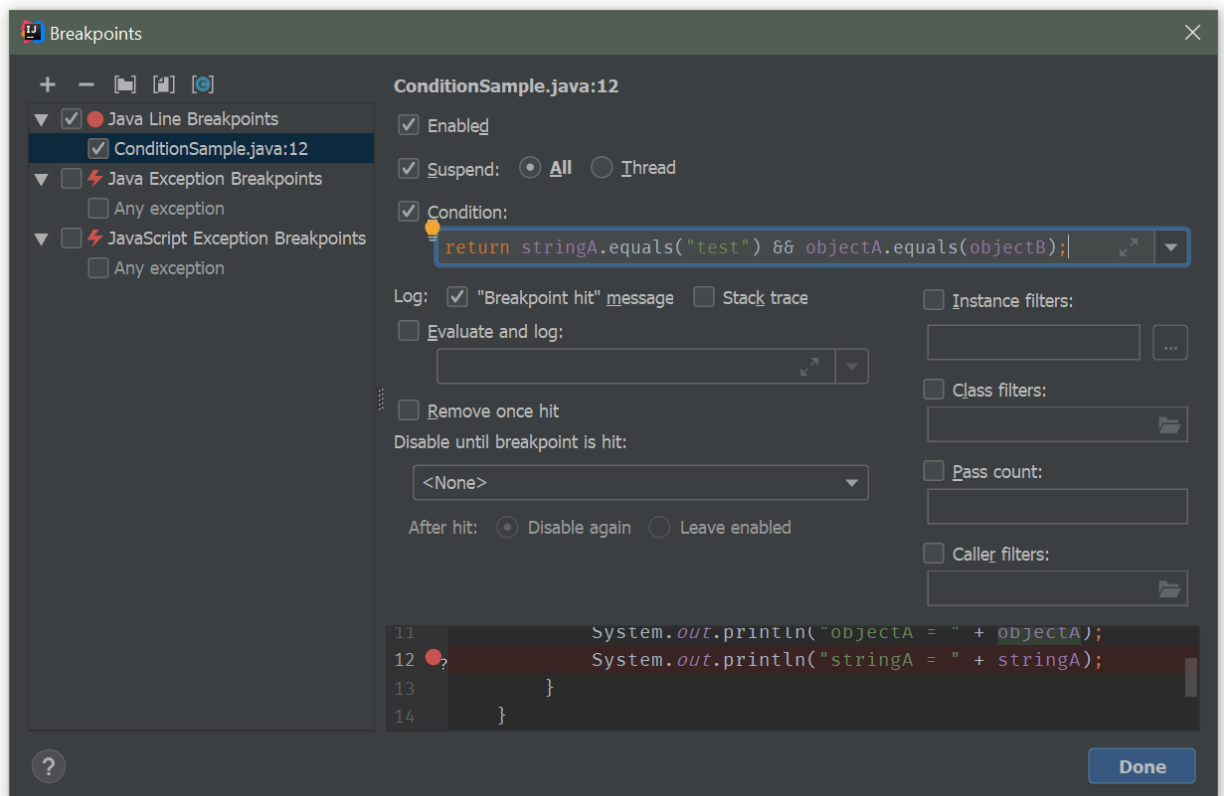




Следующая инструкция условия также будет работать:

```
return stringA.equals("test") && objectA.equals(objectB);
```

Обратите внимание, что существует известная проблема, в которой после условия, указывающего ожидается, что точка с запятой. Если вы добавите точку с запятой, условие станет недействительным, и вам также придется добавить оператор `return`, чтобы снова сделать его действительным. Это косметическая проблема, и вы можете либо использовать условие без точки с запятой и игнорировать ошибку, либо добавить точку с запятой и `return`, чтобы сделать его допустимым:



Заключение

В заключении конечно же хочется сказать о том, что программная отладка – очень важный инструмент в жизни программиста. Понимание, как пользоваться таким инструментом требуется при устройстве на работу. Если вы умеете пользоваться программным отладчиком, то никаких проблем не составит процесс нахождения ошибки. Дебаггер – очень полезный инструмент, который очень просто изучить!